nMOLDYN: User's Guide

Vania Calandrini[a], Paolo Calligari[a], Konrad Hinsen[b,c], Gerald R. Kneller[b,c] [1]

[a]Institut Laue-Langevin 6, rue Jules Horowitz BP 156 - 38042 Grenoble Cedex 9, France
[b]Centre de Biophysique Moleculaire, CNRS Rue Charles Sadron, 45071 Orleans, Cedex 02, France
[c]Synchrotron Soleil - Division Experiences, Saint Aubin - BP 48 91192 Gif sur Yvette Cedex, France

December 19, 2006

[1]Corresponding author. Electronic mail:kneller@cnrs-orleans.fr

Abstract

*n*MOLDYN is a modular program package for the analysis of Molecular Dynamics trajectories, especially designed for the computation and decomposition of neutron scattering spectra. The current release 3.0 of *n*MOLDYN is an upgrade of the version *n*MOLDYN 2.0, which extends the functionality of the original version *n*MOLDYN 1.0 (G.R. Kneller, V. Keiner, M. Kneller, M. Schiller Comp. Phys. Comm. 91, 191-214 (1995)), and provides a much more convenient user interface (both graphical/interactive and batch), and can be used as a tool set for implementing new analysis modules. This was made possible by the use of a high-level language, Python, and of modern object-oriented programming techniques. *n*MOLDYN allows one to calculate the mean-square displacement, the velocity autocorrelation function as well as its Fourier Transform (the density of states) and its memory functions, the angular velocity autocorrelation function and its Fourier transform, the reorientational correlation function. Moreover it can compute several quantities related to neutron scattering: the coherent and incoherent intermediate scattering functions with their Fourier transforms and their memory functions, and the elastic incoherent structure factor. Additionally, the *n*MOLDYN package allows one to construct modified trajectories from an input trajectory ; rigid-body trajectories, in which the internal motions of the molecules (or parts thereof) are eliminated, angular trajectories, which describe rigid-body motions by center-of-mass and orientational (quaternion) coordinates, frequency-filtered trajectories, from which motions outside a specified frequency interval are eliminated. As nMOLDYN 2.0, nMOLDYN 3.0 use the Molecular Modelling Toolkit (MMTK), an Open Source program library for molecular simulation applications, and expects trajectories to be in MMTK format (netCDF files). Trajectory converters for CHARMM/X-PLOR/NAMD, DLPOLY and AMBER trajectories are included in the distribution of *n*MOLDYN. All nMOLDYN calculations can be applied to a whole system or to arbitrary subsets. The most common subsets can be selected in the graphical interface from a list, less common selections can be specified by Python code via the command-line interface. *n*MOLDYN 2.0 was written by Tomasz Rer, Krzysztof Murzyn, and Konrad Hinsen and the upgrade, *n*MOLDYN 3.0, by Paolo Calligari.

Contents

List of Figures

Introduction

Although Molecular Dynamics simulation techniques are widely used in physics, chemistry and biology, their possibilities are often not fully exploited because of the lack of easy-to-use analysis tools. This is especially true for very complex systems which force most computational scientists to use standard program packages containing only very limited functionality to analyze MD trajectories.

In this NOTE we present the new release 3.0 of the modular program package nMOLDYN, which replaces the original version nMOLDYN 2.0, for the analysis of Molecular Dynamics trajectories.

The program nMOLDYN was developed mainly for use in connection with neutron scattering experiment, although many of the quantities are also used in other contexts. The combination of neutron scattering experiments and molecular dynamics (MD) simulations is a powerful tool to study the structure and dynamics of complex molecular systems. Neutron scattering is sensitive to time and space correlations of atomic positions on the ns time scale and the Å length scale [1, 2]. These are exactly the time and space domains covered by classical molecular dynamics simulations. On the length scale under consideration the neutron-target interaction can be modelled by pseudopotentials with zero range which are centered on the atomic nuclei of the targets. The coupling between neutron and target is described by so-called scattering lengths describing the strength of the neutron-nucleus interaction [1]. The differential scattering cross section can be expressed in terms of quantum time correlation functions of the spatially Fourier transformed particle density. The corresponding *classical* time correlation function can be easily obtained from molecular dynamics simulations. This enables a direct comparison between simulated and measured neutron scattering intensities for classical systems if recoil effects in the scattering process are not dominant [3]. The experimental data can be used to test the quality of the MD force field which is the central input for the simulations [4, 5, 6, 7]. Conversely, the simulated intensities allow a detailed analysis of the dynamical and structural behaviour of the system under consideration [8, 9]. The latter is particularly important for complex systems for which an interpretation of the measured intensities in terms of simple analytical models is difficult, if not impossible.

The program package nMOLDYN allows neutron scattering intensities to be efficiently calculated from MD simulations. The calculation of various space and time correlation functions permits a detailed analysis of the structure and

dynamics of the system under consideration. *n*MOLDYN contains modules for the calculation of incoherent and coherent dynamic structure factors, elastic incoherent structure factors (EISFs), mean-square displacements, translational velocity autocorrelation functions, and memory functions. In addition rigid body trajectories of subunits of the system can be extracted from molecular dynamics trajectory files. These subunits can be arbitrarily defined, their size can range from a few atoms to a whole domain in a macromolecule. From the rigid body trajectories angular correlation functions and reorientational correlation functions can be obtained.

The second generation *n*Moldyn presented here, offers an interactive graphical user interface for standard calculations, highly flexible script-based processing for non-standard applications and a machine-independent compact binary file format. These improvement were made possible by the use of

1. the object oriented high-level Python [10];
2. the fast array package Numerical Python;[11]
3. the efficient FFT implementation FFTW;[12]
4. the portable binary file format netCDF and the corresponding library;[17]
5. the scientific computing library Scientific Python;[18]
6. the molecular simulation library MMTK;[19]

All of these packages are developped and distributed following the Open Source principles [20]; anyone can use and improve them without being hindered by licensing restricitons. All the time-consuming algorithms use efficient implementations in C or Fortran, which are provided by other packages (Numerical Python, FFTW, and MMTK) together with a Python interface, while *n*Moldyn itself contains only Python modules.

This NOTE is organized as follows: Section 1 contains a description of the requirements for *n*MOLDYN installation and a description of the installation procedure. Section 2 contains a description of the different options and quantities that can be calculated with *n*MOLDYN, described in the order in which they appear within the graphical interface. Section 3 contains the definitions and the theoretical background of the various quantities provided by *n*MOLDYN. In Section 4 the algorithms for the numerical calculation of the various quantities are discussed.

Chapter 1

Installing nMOLDYN

The current version of nMOLDYN is 3.0.1. The nMOLDYN distribution contains the full Python source code, which is covered by the CeCILL License.[34]

1.1 Linux users

If your Linux distribution uses the RPM package manager, use the source RPM file to build a binary RPM file compatible with your system. The command is

```
rpm --rebuild nMOLDYN-3.0.1-1.src.rpm
```

Then install the resulting binary RPM file.

The source RPM file of nMOLDYN can be downloaded from the site:
http://dirac.cnrs-orleans.fr/nMOLDYN/download.html

Before compiling and installing nMOLDYN please make sure the following packages are installed and working correctly.

1. the netCDF library
2. the Scientific Python
3. the MMTK library
4. the FFTW library
5. the Python interface to FFTW

If you don't have them, you can get the corresponding source RPM file from the site: *http://dirac.cnrs-orleans.fr/nMOLDYN/download.html* , although these are not necessarily the most recent versions available.

For installation, compile and install the packages one by one in the order given above, followed by the source RPM for nMOLDYN. It is important to

install the binary RPM for each package before building the binary RPM for the following one.

If your Linux distribution does not use RPM, install from the tar file. The tar file of nMoldyn can be downloaded from the site
http://dirac.cnrs-orleans.fr/nMOLDYN/download.html
The instructions are

```
tar xzf nMOLDYN-3.0.1.tar.gz
cd nMOLDYN-3.0.1
python setup.py build
python setup.py install
```

The last command may require administrator privileges.
Before installing nMoldyn, please check the list of prerequisites below.

1. the Python interpreter (version 2.2 or higer)[35]
2. the Tk library
3. the netCDF library [36]
4. Numeric Python (recommended version, Numeric 23.8.2)[18]
5. Scientific Python [37]
6. MMTK (version 2.2 or higher)[16]

Python itself is part of all modern Linux distributions, as is Tk. Numerical Python, is provided as an optional package with most Linux versions (usually called python-numeric). Furthermore, it is advisable to install the FFTW library [12] and its Python interface to speed up the FFT operations inside *n*MOLDYN. *n*MOLDYN uses FFTW automatically if it is installed, and the FFTPACK routines from Numerical Python otherwise. Please, compile and install the packages one by one in the order given above, followed by *n*MOLDYN. To dowload the tar files of all of the prerequisites, see the sites cited in the corresponding references.

1.2 Macintosh users

There are ready-made installers for everything you need. Please install these packages in the indicated order:

1. MacPython 2.4.1

2. netCDF 3.6.0
3. TclTkAqua 8.4.9
4. Numeric 23.7 (from http://pythonmac.org/packages/)
5. ScientificPython 2.4.9
6. MMTK 2.4.2
7. nMOLDYN 3.0.1

You then have a clickable application called "nMOLDYN" in your Applications folder and the various command line tools (pMoldyn, xMoldyn, dcd_to_nc, nc_to_dcd, dlpoly_to_nc) /usr/local/bin. If you use these command line tools, make sure that /usr/local/bin is in your shell's search path.

It is of course also possible to install nMOLDYN from the source code distribution as for other Unix systems.

To download all of the packages above, see the site *http://dirac.cnrs-orleans.fr/nMOLDYN/download.html*.

Chapter 2

Using nMOLDYN

nMoldyn provides two user interface to accomodate a wide range of users,

- Graphical Interface: xMoldyn
- Command-Line Interface: pMoldyn

and expects input trajectories to be in netCDF format.

2.1 Trajectories

nMOLDYN expects trajectories to be in netCDF format and follow the conventions of the Molecular Modeling Toolkit (MMTK). Trajectories that have not been produced with MMTK or MMTK-based programs must be converted to MMTK format before they can be analyzed with nMOLDYN. This conversion is necessary because no other common trajectory format permits efficient access both to conformations at a given time and to one-atom trajectories for all times. In addition to providing such an access, the netCDF format has several advantages that make it particularly suitable for archiving trajectories:

- compact files (binary storage)
- machine-independent format
- fully self-contained, complete information about the system is stored in the trajectory file.

The conversion of the trajetories from the DLPOLY format to the MMTK format can be made directly via the nMOLDYN graphical interface (see paragraph 2.2.1). Otherwise nMOLDYN comes with two converters for foreign formats.

- **dcd_to_nc** converts from the DCD format used by the programs CHARMM, X-PLOR, and NAMD. In addition to the DCD file, a compatible PDB

is required as a system definition. Note that DCD files are machine-dependent, the converter accepts only files that are compatible with the machine it runs on. There is also an inverse converter, **nc_to_dcd**.

- **dlpoly_to_nc** converts from the DLPOLY format. It reads the DLPOLY files FIELD and HISTORY in order to produce a trajectory file in netCDF format.

Run any of these converters without input arguments in order to get usage instructions. An elaborate converter for AMBER trajectory files is available separately.

Many netCDF utilities, and various visualization and analysis packages to access netCDF data are currently available. For an up-to-date list of freely-available and commercial software that can access or manipulate netCDF data, see the NetCDF Software list in Ref. [13]. Here we recall the ncdump and ncgen utilities. These two tools convert between binary netCDF files and a text representation of netCDF files. ncdump generates an ASCII representation of a specified netCDF file on standard output. The ASCII representation is in a form called CDL (network Common Data form Language) that can be viewed, edited, or serve as input to ncgen. A CDL description consists of three optional parts: dimensions, variables, and data. The dimentions part contains the shapes of one or more of the multidimensional variables. The variable part may contain variable declarations (name, data type, and shape of a variable) and attribute assignments (units, special values, maximum and minimum valid values, and packing parameters). See Ref. [14] for a description of the CDL form. The output from ncdump is intended to be acceptable as input to ncgen which can generate a binary netCDF file from a CDL file. Moreover the ncgen tool can generate a C or FORTRAN program that creates a netCDF file. For a description of the ncdump and ncgen utilities see Ref. [14]. Here we recall some useful Unix comands for invoking ncdump as a simple browser for netCDF trajectory file, to display the dimension names and sizes; variable names, types, and shapes; attribute names and values; and optionally, the values of data for all variables or selected variables:

```
ncdump [ -c | -h] [-v var1,...]  [input-file]
```

where:

`-c`

Show the values of coordinate variables (variables that are also dimensions) as well as the declarations of all dimensions, variables, and attribute values. Data values of non-coordinate variables are not included in the output. This is often the most suitable option to use for a brief look at the structure and contents of the netCDF file.

`-h`

Show only the header information in the output, that is, output only the declarations for the netCDF dimensions, variables, and attributes of the input file,

but no data values for any variables. The output is identical to using the '-c' option except that the values of coordinate variables are not included. (At most one of '-c' or '-h' options may be present.)

`-v var1,...`
The output will include data values for the specified variables, in addition to the declarations of all dimensions, variables, and attributes. One or more variables must be specified by name in the comma-delimited list following this option. The list must be a single argument to the command, hence cannot contain blanks or other white space characters. The named variables must be valid netCDF variables in the input-file. The default, without this option and in the absence of the '-c' or '-h' options, is to include data values for all variables in the output.

2.2 Graphical Interface: xMoldyn

Through the *n*Moldyn graphical interface, the user first open a trajectory, then specifies the parameters for the calculation he wishes to perform and finally starts the calculation itself. Moreover he can check the status of running calculations and inspect their results. The graphical interface gives access to most of the functionality of *n*Moldyn. The only limitation concerns the selection of the part(s) of the full system that will be used in a particular calculation. Actually, only the most common selections can be made: small molecules can be selected collectively (e.g., all water molecules), and macromolecules can be selected individually, allowing in addition the restriction to certain subparts (e.g. only the backbone). The same options exist to specifying which parts of the system should be deuterated. Alternatively, a detailed selection can be made by marking the atoms to be used in a PDB file. Moreover, from the graphical user interface it is possible to create an input file for the command-line interface. Such input file provides a convenient starting point for customization.
To run the interactive program based on the graphical interface, type

```
xMoldyn
```

The start-up window of *n*Moldyn will pop-up (see Fig.2.1). This window contains five drop down menu buttons,

- **File**
- **Dynamics**
- **Scattering**
- **View**
- **Help**

three input text fields,

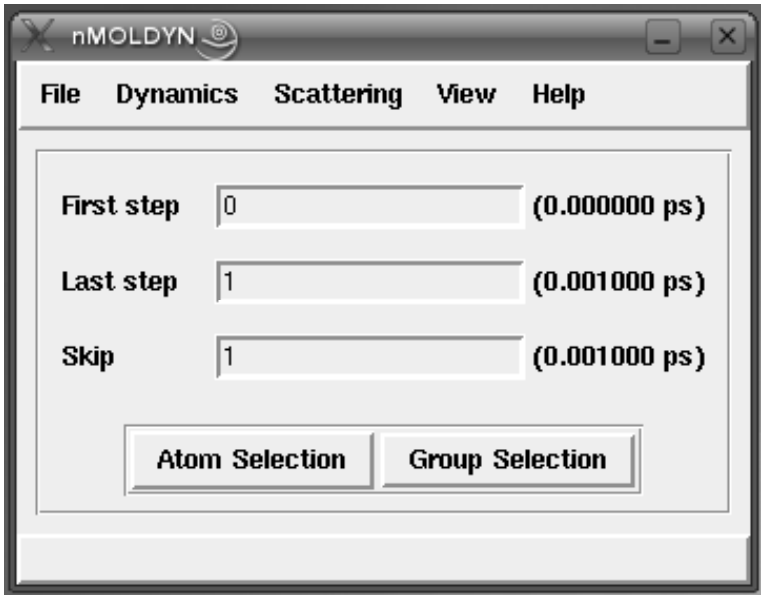

Figure 2.1: Start-up window of *n*Moldyn

- **First Step**
- **Last step**
- **Skip**

and two buttons,

- **Atom selection**
- **Group Selection**

All inputs concerning the system are set via the drop down menu button **File**, the three text fields **First Step, Last Step, Skip** and the two auxiliary buttons **Atom Selection** and **Group Selection**. The drop down menu buttons **Dynamics** and **Scattering** allow one to start the calculation of the quantities one is interested in. The **View** and **Help** buttons allow one to inspect input/output data and ask for help respectively. All of these functions will be described below.

The following notation is used throughout this section: N is the number of (selected) atoms in the system, or in the subsystem for which the analysis is performed. The atoms are labelled by Greek indices α, β, ... Their positions are denoted by $\mathbf{R}_\alpha$ and their velocities by $\mathbf{v}_\alpha$. The coherent and incoherent scattering lengths are defined by

$$b_{\alpha,\mathrm{coh}} = \overline{b_\alpha} \tag{2.1}$$

$$b_{\alpha,\mathrm{inc}} = \sqrt{\overline{b_\alpha^2} - \overline{b_\alpha}^2} \tag{2.2}$$

respectively, where the symbol denotes an average over isotopes and relative spin orientations of neutrons and nucleus. In quantities that are averages over all atoms, w_α, denotes a set of weights. nMoldyn implements different weighting schemes for the atoms:

- Equal weighting:
$$w_\alpha = \frac{1}{N}$$

- Incoherent neutron scattering:
$$w_\alpha = \frac{b_{\alpha,\mathrm{inc}}^2}{\sum_\alpha b_{\alpha,\mathrm{inc}}^2}$$

- Coherent neutron scattering:
$$\sqrt{w_\alpha} = \frac{b_{\alpha,\mathrm{coh}}}{\sqrt{\sum_\alpha b_{\alpha,\mathrm{coh}}^2}}$$

- Mass weighting:
$$w_\alpha = \frac{m_\alpha}{\sum_\alpha m_\alpha}$$

where $\sum_{\alpha=1}^{N} w_\alpha = 1$.

For the calculation of the velocity autocorrelation function starting from the atomic coordinates and for the calculation of the angular autocorrelation function starting from the quaternion parameters, nMoldyn performs a numerical differentiation of the input data. The program implements the following differentiation schemes:

- Fast
- order 2
- order 3
- order 4

- order 5

Using option *fast*, the first time derivative of each point $r(t_i)$ is calculated as

$$\dot{r}(t_i) = \frac{r(t_{i+1}) - r(t_i)}{\Delta t}, \tag{2.3}$$

where Δt is the time step. Choosing option *order* N with $N = 2, ..., 5$, nMoldyn calculates the first time-derivative of each point $r(t_i)$ $(r = x, y, z)$ using the N-order polynomial interpolating the $N+1$ points across $r(t_i)$, where $r(t_i)$ belongs to this set [15]. The user can select the suited differentiation scheme.

2.2.1 Input: File

Pressing the button **File** brings up a menu from which it is possible to choose the following options:

- Open trajectory
- Open trajectory set
- Open a DLPOLY trajectory
- Open scattering function
- Write PDB file
- Show running calculations
- Quit

The **Open trajectory** option allows one to select the **netCDF** file of the trajectory which will be used to perform the calculations. Clicking it a file/directory selection window like that of Fig.2.2 pops up, listing all the sub-directories and the files in the working directory. To select a directory, double click it. By doing this, the complete pathname of the selected directory appears on the **directory** selection button at the bottom of the window. To select the parent directory left-click the directory selection button. When a file in the list is highlighted (clicked) its name is shown in the **file name** field. The selection is confirmed by pressing the button **Open** of the file/directory selection window. By doing this, the file/directory selection window disappears.

The **Open trajectory set** option allows a *trajectory set* file to be selected as input file. The trajectory set file permits to treat a sequence of trajectory files like a single trajectory for reading data. The trajectory files must all contain data for the same system. The variables stored on the individual files need not be the same, but only variables common to all files can be accessed (see pag. 141 of the MMTK manual [16] for more details about the construction of these files). The selection of a *Trajectory Set* file can be made via a file/directory selection window like that of Fig.2.2.

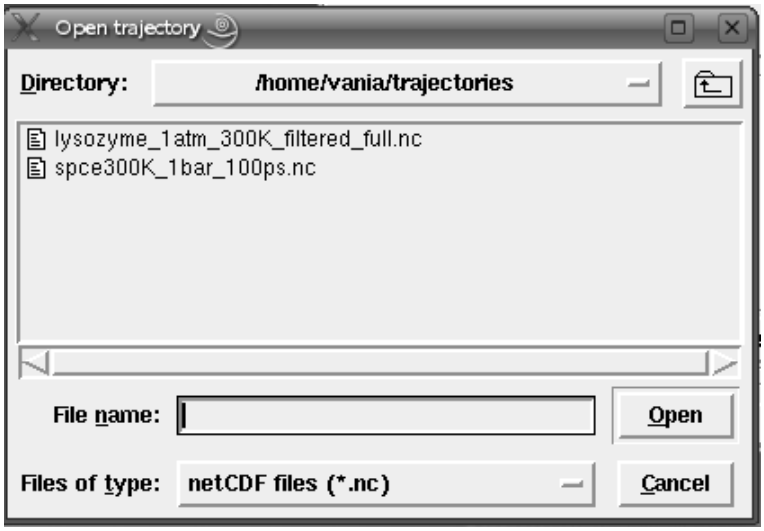

Figure 2.2: file-directory selection window

The**Open DLPOLY trajectories** button allows a DLPOLY trajectory to be selected and converted in netCDF format. The selection is made through a file directory selection window like that of Fig.2.2. The new netCDF file of the trajectory can then be selected as input file from the **Open trajectory** button.

The option **Open Sacttering Function** allows one to open an output netCDF file previously calculated in order to display it via the **Display** button in the **View** menu.

The option **write PDB file** allows a PDB file to be written starting from a netCDF trajectory. The positions of all of the atoms of the system at a given time step of the simulated trajectory can be stored in a PDB file. At first the trajectory has to be selected via the **Open Trajectory** or the **Open Trajectory set** buttons. The user can then select the desired time step by inserting the corresponding integer value in the **First Step** field and then pressing enter. Finally the corresponding PDB file can be created via the **write PDB file** button.

Once a calculation is running, the user can check its progress by selecting the**Show running calculations** option. A window showing in real time job progress, time, and date, will pops up see Fig.2.3.

The **Quit** button will terminate the program.

2.2.2 Input: First Step, Last Step, Skip

Once the trajectory has been selected, the user can to specify the starting time-step, the last time-step, and the number of time-steps which have to be skipped from the input MD trajectory to perform the calculation. The above values

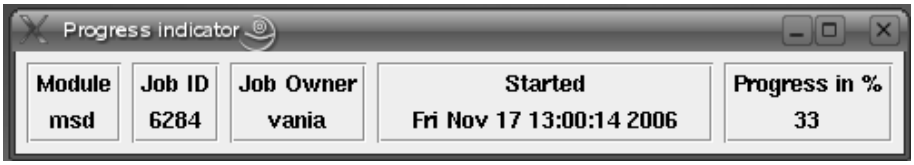

Figure 2.3: Job progress

have to be entered in the text fields named **First Step**, **Last Step**, and **Skip**, respectively. The default values are: 1, N (with N equal to the total number of time steps contained the selected trajectory) and 1. This means that the program takes by default the first and the last point of the input MD trajectory as first and last time step for the calculation of the required quantity and that every time step is extracted from the input trajectory (i.e. all of the time steps of the input trajectory are taken into account [skip=1]). Any other choice can be made by replacing the above default values by new integer values and then pressing 'Enter' in every text field. The corresponding time values expressed in units of *ps* are shown on the right side of the text fields.

2.2.3 Input: Atom Selection, Group Selection

Once the trajectory of the system has been selected, the buttons **Atom selection** allows one to select all or part(s) of the full system that will be used in a particular calculation and to specifying which parts of the system should be deuterated. The selection will be used to calculate:

Dynamics

- mean-square displacement
- velocity autocorrelation function
- density of states

Scattering

- coherent scattering function
- coherent scattering AR analysis
- incoherent scattering function (Gaussian approximation)
- incoherent scattering function
- incoherent scattering AR analysis
- EISF

Pressing the **Atom Selection** button a window nemed *Atom selection* pops-up. This window contains a input field named **Read from PDB file**, having an auxiliary button **Browse**, and a button named ***my system***. A detailed selection of the system can be made by marking the atoms to be used in a PDB file which will be imported via the **Browse** button. The selection is made by replacing the atom label in the last column of the PDB file by '*'. The user can specify in the same PDB file possible deuteration of part(s) of the system or of the overall system. Alternatively, most common selections can be made via the button ***my system***. In case of trajectories of simple atomic or molecular systems, the button ***my system*** allows one to take into account all of the atoms or molecules of the simulation, while in case of more complex systems, containing for example proteins, it allows the single macromolecule to be selected (see Fig.2.4 and Fig.2.5; here the button and the window *my system* become respectively *Water* and *Protein.0*). Pressing the ***my system*** button, a new window *my system* pops up allowing the user to specify which atoms or groups have to be taken into account for the calculations and whethever the system, or part of it, should be deuterated. The first column lists the atoms or the groups which can be selected and the second column lists the atoms or the groups which can be deuterated. Fig.2.4 and Fig.2.5 show as an example the possible choices listed in the window *my system* in the case of water and in the case of a protein.

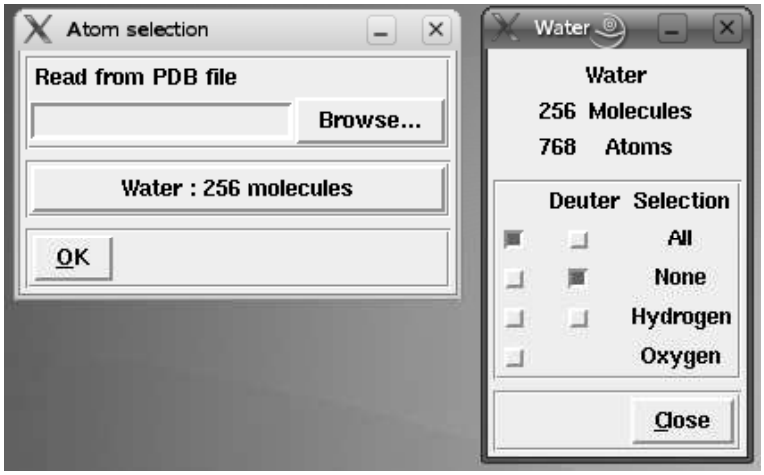

Figure 2.4: Atom selection in water

Pressing the **close** button the choice is confirmed and the window *my system* is closed. Pressing the **OK** button in the *my system* window the user confirms the choice of the system.

To analyze the dynamics of complex molecular systems it is often desirable to consider the overall motion of molecules or molecular subunits, i.e. their

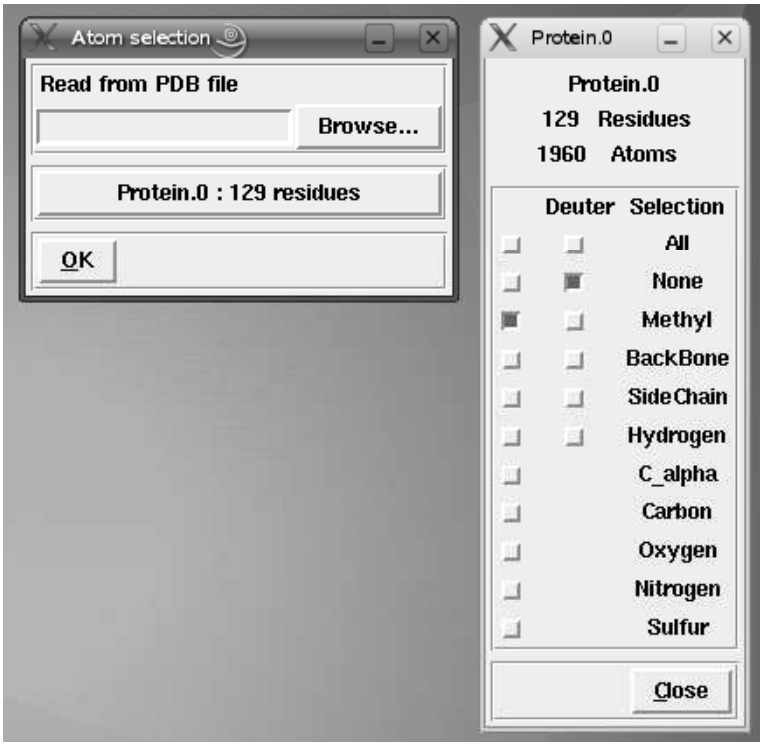

Figure 2.5: Atom selection in lysozyme

rigid-body motion. The **Group Selection** button allows the user to select - within the system already selected by the **atom selection** button - a particular atomic group that will be used to calculate some functions describing its rigid-body dynamics:

Dynamics

- angular trajectory
- rigid body trajectry
- rigid body rotation trajectory
- angular velocity autocorrelation function (VACF)
- spectrum of angular VACF
- reorientational correlation function

The functions describing the rigid body dynamics have not been fully tested yet, but they will be soon available. At present the guide

contains only a description of the theoretical background and of the relevant algorithms.

Once the trajectory has been imported, pressing the **Group Selection**, a window named *Groups*, listing all the possible groups which can be taken into account to perform the above calculations, pops up. For example small molecules can be selected collectively (e.g., all water molecules), and macromolecules can be selected individually. This option is very useful when one deals with complex systems, containing for example proteins. In this case, a drop-down menu button relevant to the individual macromolecule appears wthin the *Groups* window. This button allows the user to select some standard groups. For example, in the case of a protein, we have the overall macromolecule, the sideChain, the backbone, the methyl groups. For each group, the user can select within the window named *selection*, wherever it has to be taken into account to perform the calculation. In addition, within the SideChain-group selection, the user can selectively mark the residues to be used (lysine, histidine, alanine, etc). Once a group has been selected in the *selection* window, the two buttons on the right become active. The first button allows one to select the *Reference* file PDB containing the group reference-position which has to be used to calculate one of the above quantities. The second button allows one to display some *information* about the reference file. Pressing the **close** button the choice is confirmed and the window is closed.

2.2.4 Output: Dynamics

Pressing the button **Dynamics** brings up a menu from which it is possible to choose the following options:

- Mean Square Displacement
- Velocity Autocorrelation Function
- Density of States
- Autoregressive Model
- Angular Trajectory
- Rigid Body Trajectory
- Rigid Body Rotation Trajectory
- Digital Filter
- Angular Velocity Autocorrelation Function
- Spectrum of angular VACF
- Reorientational Correlation Function

In this section we give a description of the dynamical quantities calculated by *n*Moldyn. For a description of the theoretical background see Chapter 3.

Mean Square Displacement (MSD)

nMoldyn allows one to computes the average mean square displacement of atoms - see Section 3.5:

$$\Delta^2(k \cdot \Delta t) = \sum_\alpha w_\alpha \Delta^2_\alpha(k \cdot \Delta t), \qquad k = 0 \ldots N_t - 1. \tag{2.4}$$

N_t is the total number of time steps in the coordinate time series. The w_α are user-defined weights which are normalized to one (see Section 2.2). One can also compute the mean-square displacement with respect to a user-defined axis:

$$\Delta^2(k \cdot \Delta t; \mathbf{n}) = \sum_\alpha w_\alpha \Delta^2_\alpha(k \cdot \Delta t; \mathbf{n}), \qquad k = 0 \ldots N_t - 1. \tag{2.5}$$

In both cases the algorithm described in Section 4.2 is applied. nMoldyn corrects the atomic input trajectories for jumps due to periodic boundary conditions. The above calculation can be carried out through the graphical interface pressing the button **Mean Square Displacement** from the drop-down menu **Dynamics**. Pressing this button will open the window *Mean-Squre Displacement* (see Fig.2.6).

The user can calculate the mean square displacement with respect to a given axis by entering in the **Projection Vector** field the components of the corresponding unit vector in the form *x,y,z*. Otherwise writing *None*, the MSD will be calculated by averaging over all the directions. In the panel **Weights** the user can select how to weight the different contributions to the MSD coming from all atoms of the system. The weights are normalized to one (see Section 2.2).

The units length for the Mean Square Displacement and the number of time steps in the output file can be selected in the panel **Length units** and **Time Steps in output** respectively. The output file is a file ascii ".plot" containing the MSD of the selected system as a function of the time in ps. The default name of the output file contains the string **MSD** followed by the name of the trajectory. The user can then change the name and/or select the destination directory through the **Browse** button. Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language (see Fig.2.7). These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of nMoldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

Velocity Autocorrelation Function (VACF)

Pressing the button **Velocity Autocorrelation Function** will open a drop-down menu button from which the user can select how to perform the calculation of the velocity autorrelation function: directly from the velocities or from the coordinates. In this case the velocities are computed by numerical differentiation of the coordinate trajectories correcting first for possible jumps due to periodic

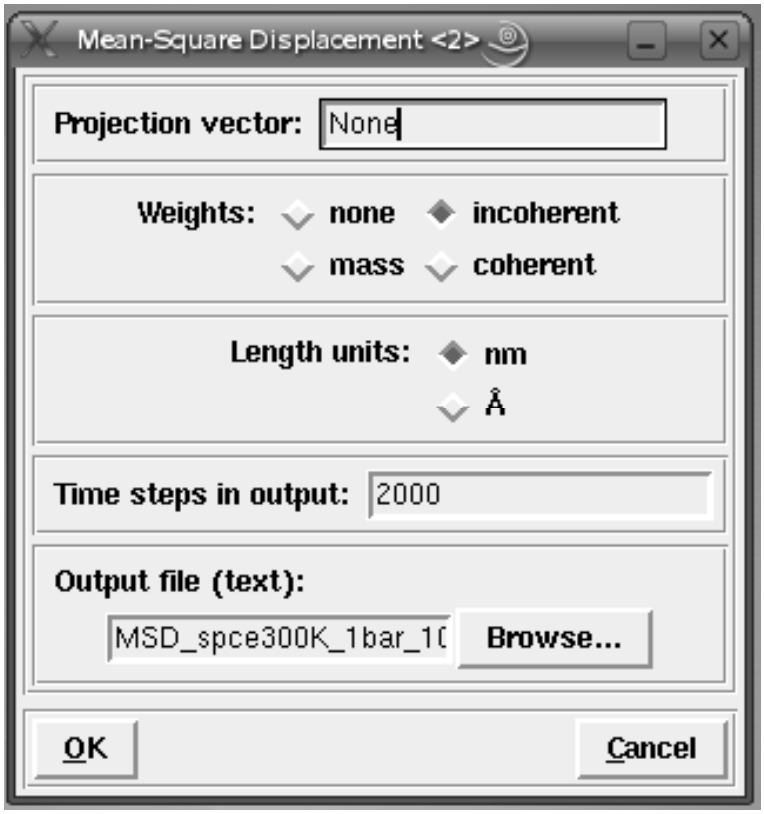

Figure 2.6: Mean Square Displacement window

boundary conditions. In both cases the program computes the discrete velocity autocorrelation function, $C_{vv}(k \cdot \Delta t)$

$$C_{vv}(k \cdot \Delta t) \doteq \sum_{\alpha} w_\alpha C_{vv;\alpha\alpha}(k \cdot \Delta t), \qquad k = 0 \ldots N_t - 1 \tag{2.6}$$

$$\tag{2.7}$$

where N_t is the total number of time steps The velocity autocorrelation functions $C_{vv;\alpha\alpha}(k \cdot \Delta t)$ is computed with the FCA-algorithm described in Section 4.1. $C_{vv}(k \cdot \Delta t)$ can be computed either for the isotropic case or with respect to a user-defined axis – see Section 3.4. The quantities w_α are user-defined weights which are normalized to one, $\sum_\alpha w_\alpha = 1$. The VACF is normalized, such that $C_{vv}(0) = 1$.

From Velocities Pressing the button **velocities** will open the window *Velocity Autorrelation Function (from velocities)* from which the user can enter some

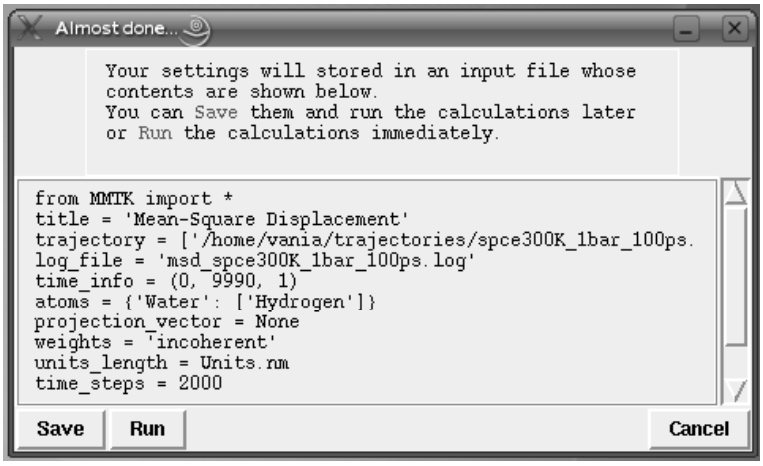

Figure 2.7: 'Almost Done' window

quantities relevant to the calculation - such as a possible projection vector, the weights, the length units, the time steps in output, the name and the location of the output file - edit the corresponding python script, close the window to come back to the start-up window of *n*Moldyn (see Fig.2.8).

To calculate the VACF with respect to a given axis, the user can enter in the **Projection Vector** field the components of the corresponding unit vector in the form *x,y,z*. Otherwise writing *None*, the VACF of each atom in the selected system will be calculated taking **v** over all directions.

In the panel **Weights** the user can select how to weight the different contributions to the VACF coming from all atoms of the system. The weights are normalized to one (see Section 2.2).

The units length for the VACF and the number of time steps in the output file can be selected in the panels **Length units** and **Time Steps in output** respectively.

The output file is a file ascii ".plot" containing the VACF of the selected system as a function of the time in ps. Since the VACF is symmetric in time, it is stored only for values on the positive time axis. The default name of the output file contains the string **VACF** followed by the name of the trajectory. The user can then change the name and/or select the destination directory through the **Browse** button.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language (see Fig.2.9). These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately throgh the button **Run**. Pressing the button **cancel** closes the window without saving settings.

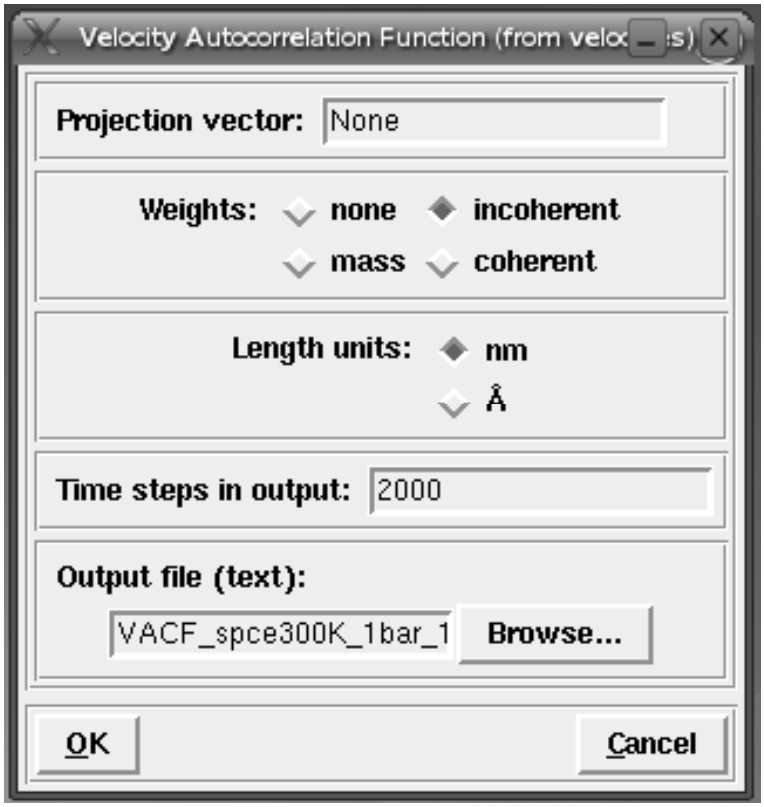

Figure 2.8: VACF from velocities

FromCoordinates Pressing the button **coordinates** will open the window *Velocity Autorrelation Function (from coordinates)*. From this window it is possible to enter the same quantities as for the calculation from velocities. In addition, the panel**differentiation** allows one to select how to perform the differentiation of the coordinate trajectories to obtain the velocities trajectory (see Fig.2.10). Refer to Section 2.2 for more details about the differentiation schemes.

Density of States (DOS)

The program calculates the discrete density-of-states (DOS), $G(n \cdot \Delta\nu)$, which is the Fourier spectrum of the VACF (see Paragraph 2.2.4):

$$G(n \cdot \Delta\nu) \doteq \sum_{\alpha} w_\alpha \tilde{C}_{vv;\alpha\alpha}(n \cdot \Delta\nu), \qquad n = 0 \ldots N_t - 1. \tag{2.8}$$

N_t is the total number of time steps and $\Delta\nu = 1/(2N_t\Delta t)$ is the frequency step. $G(n \cdot \Delta\nu)$ can be computed either for the isotropic case or with respect to a

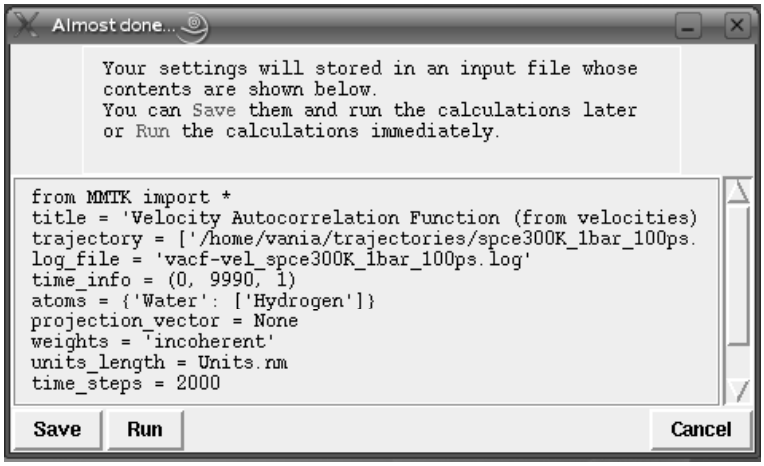

Figure 2.9: VACF from velocities. Python script

user-defined axis – see Section 3.4. The quantities w_α are user-defined weights which are normalized to one, $\sum_\alpha w_\alpha = 1$. The spectrum $G(n \cdot \Delta\nu)$ is computed from the *unnormalized* VACF, such that $G(0)$ gives an approximate value for the diffusion constant $D = \sum_\alpha D_\alpha$ – see Eqs. (3.39) and (3.40). $G(n \cdot \Delta\nu)$ is smoothed by applying a Gaussian window in the time domain [38]- see Section 4.1. Its width in the time domain is $sigma_t = alpha/T$, where T is the length of the simulation. We remark that the diffusion constant obtained from DOS is biased due to the spectral smoothing procedure since the VACF is weighted by this window Gaussian function. nMoldyn computes the density of states starting from both atomic velocities and atomic coordinates. In this case the velocities are computed by numerical differentiation of the coordinate trajectories correcting first for possible jumps due to periodic boundary conditions.

From velocity Pressing the button **velocity** will open the window *Density of states (from velocities)* (see 2.11).

To calculate the DOS with respect to a given axis, the user can enter in the **Projection Vector** field the components of the corresponding unit vector in the form *x,y,z*. Otherwise, writing *None*, the DOS will be isotropically calculated.

In the panel **Weights** the user can select how to weight the different contributions to the DOS coming from all atoms of the system (see Section 2.2). The panel **window width for FFT: percento of trajectory lenght** allows one to select the width of the Gaussian function to be used in the smoothing procedure for the calculation of the DOS - see Section 4.1. The width is defined with respect to the lenght of the trajectory.

The length units for the DOS, the frequency units and the number of points in the output spectrum can be selected in the panels **Length units**, **Frequency units**, and **Points in spectrum**, respectively.

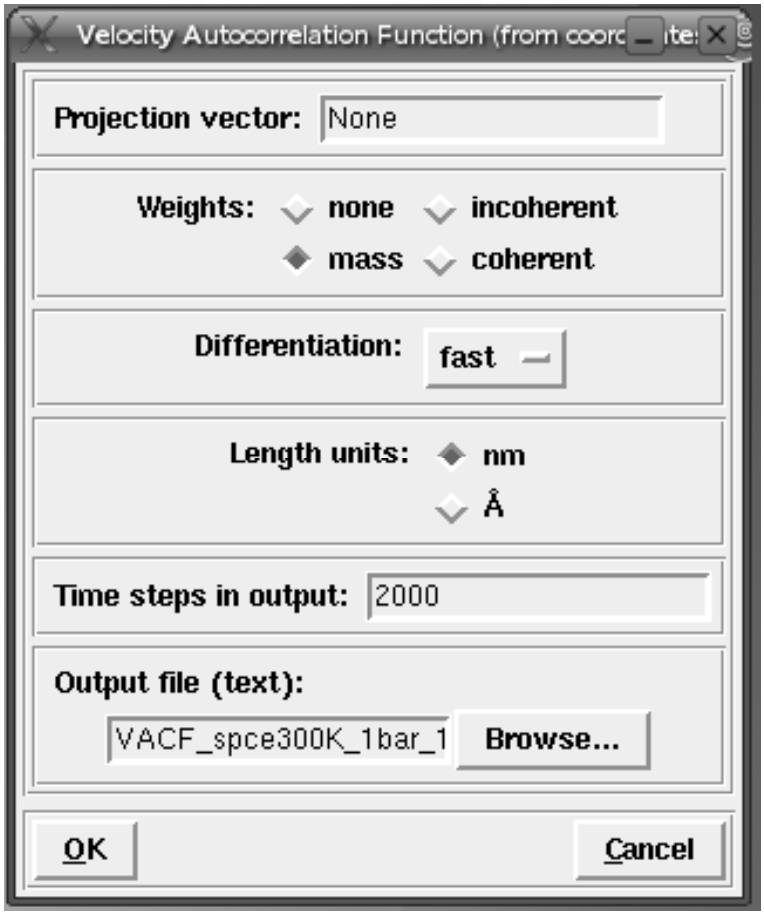

Figure 2.10: VACF from coordinates

The output file is a file ascii ".plot" containing the DOS of the selected system as a function of the frequency in the selected units. Since the VACF is symmetric in time, its Fourier spectrum is stored only for values on the positive frequency axis. The default name of the output file contains the string **DOS** followed by the name of the trajectory. The user can then change the name and/or select the destination directory through the **Browse** button.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language (see Fig.2.12). These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

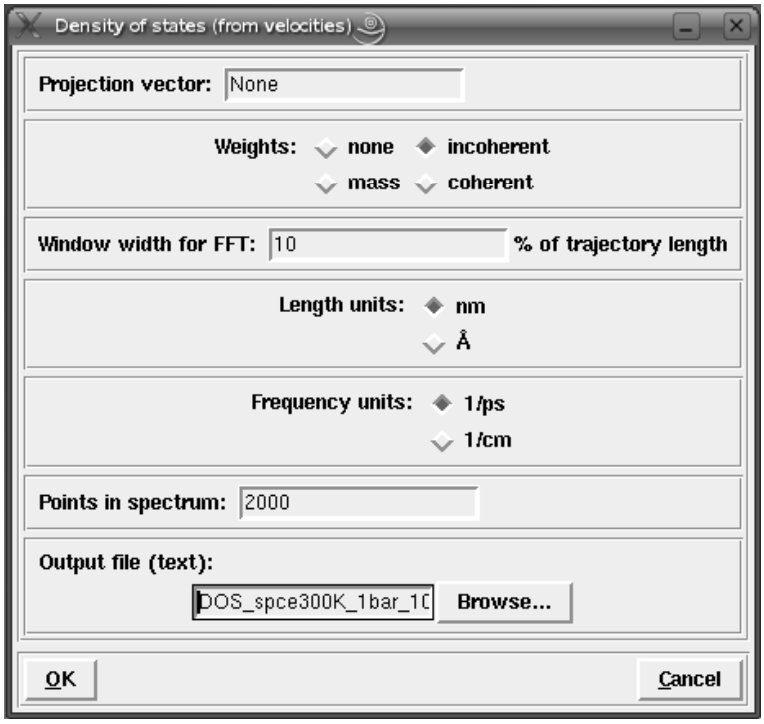

Figure 2.11: DOS from velocities

From coordinate Pressing the button **coordinates** will open the window *Density of states (from coordinates)*. From this window it is possible to enter the same quantities as for the calculation of the DOS from velocities. In addition, the panel**differentiation** allows one to select how to perform the differentiation of the coordinates trajectories (see Fig.2.13) to obtain the velocities trajectory. Refer to Section 2.2 for more details about the differentiation schemes.

Autoregressive model (AR)

In the framework of the autoregressive model, *n*Moldyn allows one to calculate the VACF, the VACF memory function, the DOS, the MSD, and the Autoregressive coefficients $a_n^{(P)}$ of the velocity trajectory, averaged over all selected atoms and three Cartesian coordinates. The user is referred to Section 3.9 and Section 4.4 for more details. Pressing the button **Autoregressive model** in the menu **Dynamics** of *n*Moldyn, a drop-down menu allows one to select how to calculate the above quantities (from coordinates or from velocities).

From velocities choosing the option **from velocities** the window **Autoregressive Model (from velocities)** in Fig 2.14 pops up.

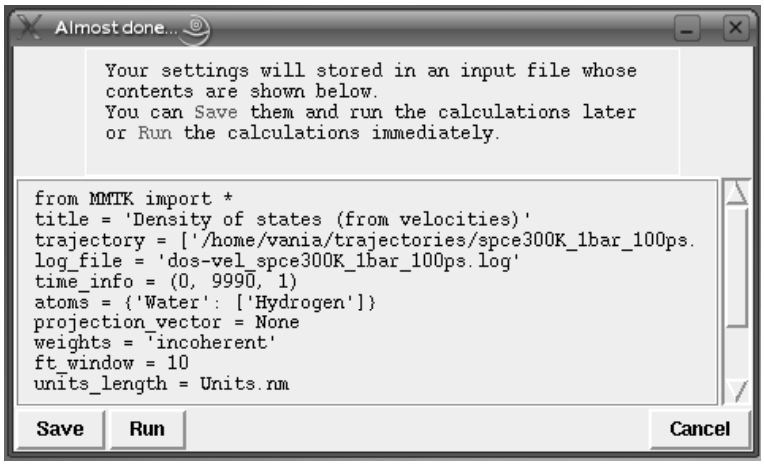

Figure 2.12: DOS from coordinate. Python script

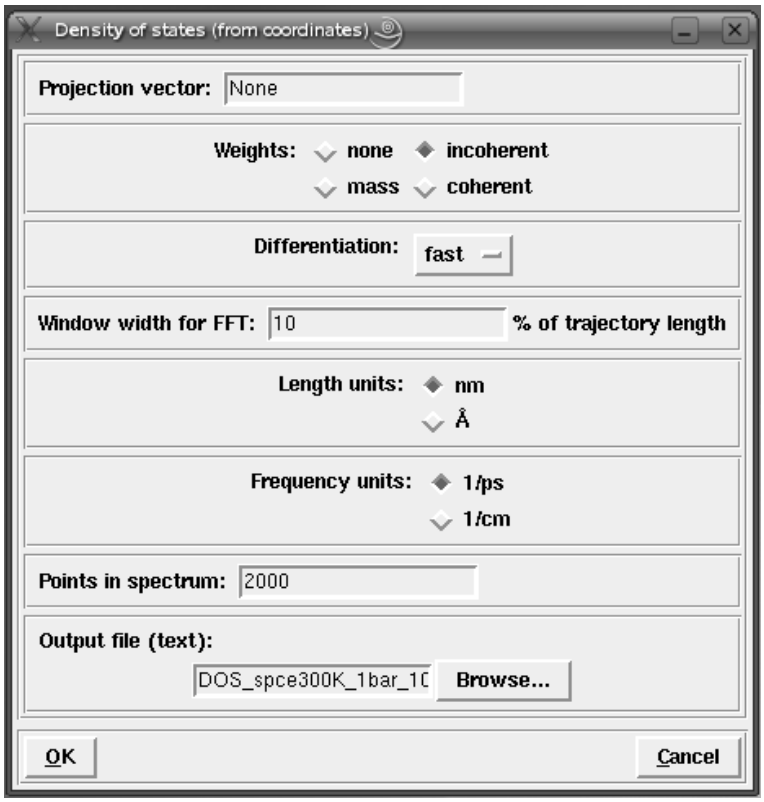

Figure 2.13: DOS from coordinates

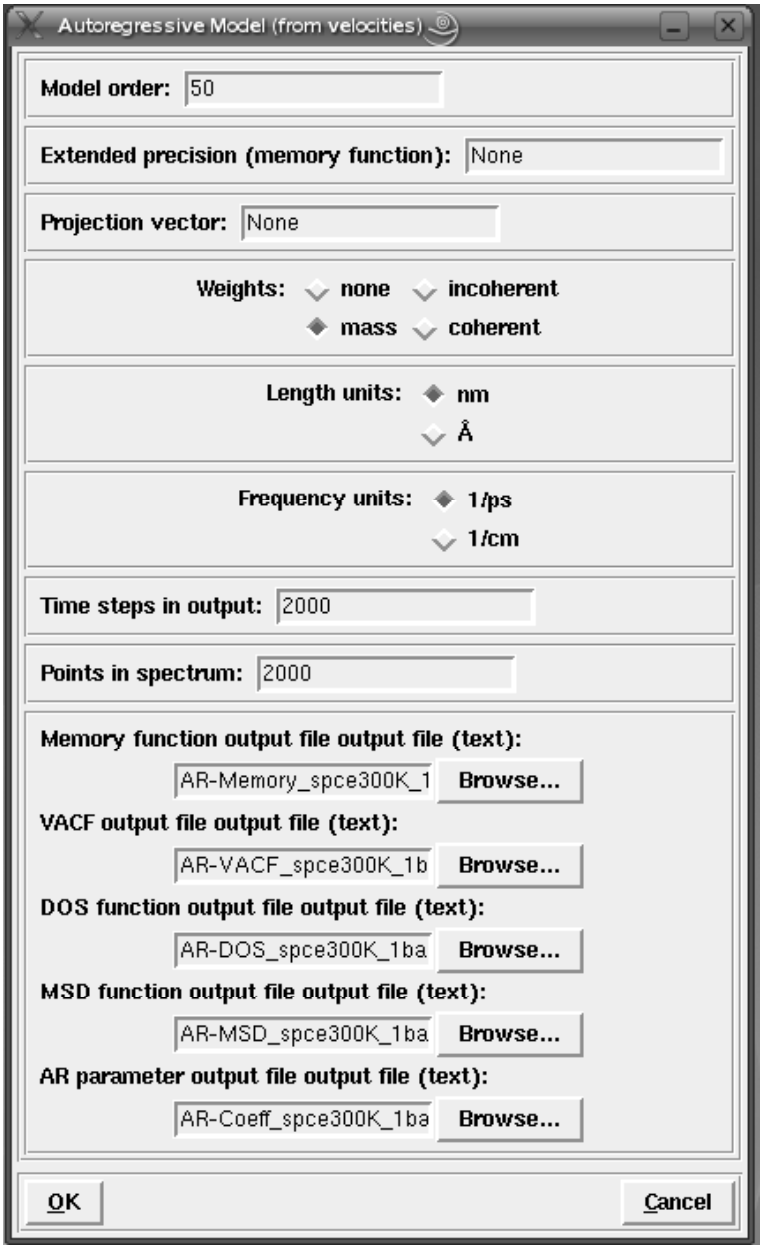

Figure 2.14: Autoregressive-model window (from velocities)

The user can enter the order (= poles number) of the Autoregressive model in the field **Model Order** (see Section 4.4). *A priori* the autocorrelation function and its power spectrum can be approximated to almost arbitrary precision by increasing the order of the autoregressive model. In practice it has been proven that reliably computation can be carried out up to P of the order of 1000 poles.

As for the *standard* calculation of VACF, DOS, and MSD, it is possible to calculate these functions isotropically or along a specific axis. This can be made through the **Projection Vector** field typing *None* or the components of the corresponding unit vector in the form *x,y,z*, respectively.

In the panel **Weights** the user can select how to weight the different contributions coming from all atoms of the system (see Section 2.2). The weights are normalized to one.

The length units, the frequency units, the time steps in the output files of VACF, MSD and VACF memory function, and the number of points in the output file of DOS, can be selected in the panels **Length units**, **Frequency units**, **Time steps** and **Points in spectrum**, respectively.

Five output files in ascii format containing the memory function of the VACF, the VACF, and the MSD, as a function of time in ps, the DOS as a function of frequency in the selected units, and the Autoregressive coefficients of the velocity trajectory (σ_P^2, σ_P, $a_n^{(P)}$), are produced. Since the VACF is symmetric in time, its Fourier spectrum is stored only for values on the positive frequency axis. The default name of the output files contain the string **AR-Memory AR-VACF**, **AR-DOS**, **AR-MSD**, **AR-Coeff**, followed by the name of the trajectory. The user can then change their names and/or select the destination directory through the **Browse** button.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

From coordinates Pressing the button **coordinates** will open the window *Density of states (from coordinates)*. From this window it is possible to enter the same quantities as for the Autoregressive model from coordinates. In addition, the panel**differentiation** allows one to select how to perform the differentiation of the coordinate trajectories to obtain the velocities. Refer to Section 2.2 for more details about the differentiation schemes.

Angular Trajectory

Option *Angular Trajectory* allows the angular trajectory of the atomic/molecular group(s), selected via the **group selection** option, to be calculated. Choosing this option *n*Moldyn performs a positional least-square fit of a set of reference structures to the corresponding set of given structures in a Molecular Dynamics trajectory. The fit algorithm is described in Section 4.3. The reference structures, stored in a pdb file, can be selected via the **group selection** option.

In general the reference structure is defined in an orthonormal body fixed coordinate system Usually this is the principal axis system in which the tensor of inertia is diagonal. For each time frame and for each group the program calculates:

- three cartesian components describing the optimal translation of the reference structure
- four normalized quaternion parameters describing the optimal rotation of the reference structure
- the fit error per site

Then the angular trajectories are calculated numerically as time integrals over the cartesian components of the angular velocity in the body-fixed frame

$$(\omega_x^{'}, \omega_y^{'}, \omega_z^{'})$$

according to

$$\Phi_i(t) = \int_0^t d\tau\, \omega_i'(\tau), \tag{2.9}$$

where the angular velocity in the body-fixed frame is related to the time derivative of the quaternion parameters via Eq. 3.42.

The output file is a file netCDF containing the angular trajectory of the selected system with respect to the rigid-body frame. The default name of the output file contains the string **AT** followed by the name of the trajectory. The user can then change the name and/or select the destination directory from the window **angular trajectory** through the **Browse** button in the *Angular trajectory* window. Pressing the **OK** button switches to the *Almost Done* window containing all input settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

Digital filtering

It is often of interest to restrict attention to motions in a specific frequency interval, both for quantitative analysis and for visualization by animated display. this is particularly useful to study low-frequency motions without being distracted by the high frequency "noise". *n*Moldyn can apply frequency filter to the atomic trajectories, either of the whole system or of a user-defined subset. The result is stored in a new trajectory file that contains only motions in the chosen interval. Frequency filtering uses a straightforward algorithm: take the discrete Fourier transform of each particle coordinate, set the Fourier coefficients outside the filtering interval to zero, and transform back to the time domain. This corresponds to applying a rectangular window in the frquency domain.

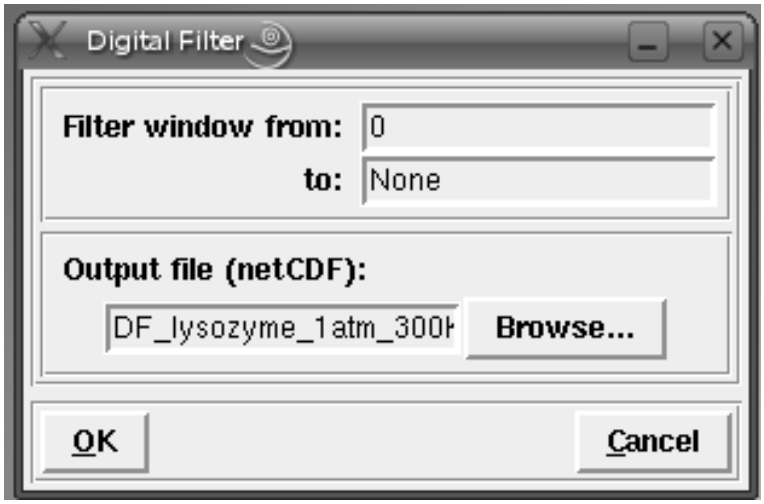

Figure 2.15: Digital Filtering window

Pressing the button **Digital Filtering** from the menu **Dynamics** of *n*Moldyn, the window in Fig.2.15 pops up The user can enter the frequency interval in THz units in the fields **Filter window**. The output file is a file netCDF containing only motions in the chosen interval. The default name of the output trajectory contains the string **AT** followed by the name of the trajectory. The user can then change the name and/or select the destination directory through the **Browse** button in the *Angular trajectory* window. Pressing the **OK** button switches to the *Almost Done* window containing all input settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

2.2.5 Output: Scattering

Pressing the button **Scattering** brings up a drop-down menu from which it is possible to choose the following options:

- Coherent Scattering Function
- Coherent Scattering AR analysis
- Incoherent Scattering Function (Gaussian Approx.)
- Incoherent Scattering Function
- Incoherent Scattering AR analysis

- EISF

In this section we give a description of the scattering quantities calculated by *n*Moldyn. For a description of the theoretical background see Chapter 3.

Coherent Scattering Function

*n*Moldyn computes the coherent intermediate scattering function on a rectangular grid of equidistantly spaced points along the time- and the q-axis, repectively:

$$\mathcal{F}_{coh}(q_m, k\cdot\Delta t) \doteq \overline{\langle\rho(-\mathbf{q},0)\rho(\mathbf{q},k\cdot\Delta t)\rangle}^{q}, \qquad k=0\ldots N_t-1,\ m=0\ldots N_q-1. \tag{2.10}$$

N_t is the total number of time steps in the coordinate time series and N_q is a user-defined number of q-shells. $\rho(\mathbf{q}, k\cdot\Delta t)$ is the Fourier transformed particle density,

$$\rho(\mathbf{q}, k\cdot\Delta t) = \sum_\alpha \sqrt{w_\alpha}\exp[i\mathbf{q}\cdot\mathbf{R}_\alpha(k\cdot\Delta t)]. \tag{2.11}$$

The symbol $\overline{\ldots}^q$ in (2.10) denotes an average over q-vectors having *approximately* the same modulus $q_m = q_{min} + m\cdot\Delta q$. The particle density must not change if jumps in the particle trajectories due to periodic boundary conditions occcur. In addition the *average* particle density, N/V, must not change. This can be achieved by choosing q-vectors on a lattice which is reciprocal to the lattice defined by the MD-box. Let $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ be the basis vectors which span the MD cell. Any position vector in the MD cell can be written as

$$\mathbf{R} = x'\mathbf{b}_1 + y'\mathbf{b}_2 + z'\mathbf{b}_3, \tag{2.12}$$

with x', y', z' having values between 0 and 1. The primes indicate that the coordinates are box coordinates. A jump due to periodic bounday conditions causes x', y', z' to jump by ± 1. The set of dual basis vectors $\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3$ is defined by the relation

$$\mathbf{b}_i\mathbf{b}^j = \delta_i^j. \tag{2.13}$$

If the q-vectors are now chosen as

$$\mathbf{q} = 2\pi\left(k\mathbf{b}^1 + l\mathbf{b}^2 + m\mathbf{b}^3\right), \tag{2.14}$$

where k, l, m are integer numbers, jumps in the particle trajectories produce phase changes of multiples of 2π in the Fourier transformed particle density, i.e. leave it unchanged. One can define a grid of q-shells or a grid of q-vectors along a given direction or on a given plane, giving in addition a *tolerance* for q. *n*Moldyn looks then for q-vectors of the form (2.14) whose moduli deviate within the prescribed tolerance from the equidistant q-grid. From these q-vectors only a maximum number per grid-point (called generically q-shell also in the anisotropic case) is kept.

The weights w_α of which the square roots appear in (2.11) are normalized to one, $\sum_\alpha w_\alpha = 1$. The density-density correlation is computed via the FCA technique described in Section 4.1.

The above calculations may be run via the graphical interface. Pressing the button **Coherent Scattering Function** brings up a drop-down menu from which the user can choose wether to calculate $\mathcal{F}_{coh}(q_m, k \cdot \Delta t)$ isotropically ($\rightarrow$ **Isotropic**), or on a given plane ($\rightarrow$ **Anisotropic**), or along a given direction ($\rightarrow$ **Explicit List**):

Isotropic Through this option, the average of Eq. 2.10 will be carried out taking on a q-shell the q-vectors whose moduli deviate within the prescribed tolerance from the required values. Pressing this button the window **Coherent Scattering Function (Isotropic)** pops-up (see Fig.2.16). The user can enter

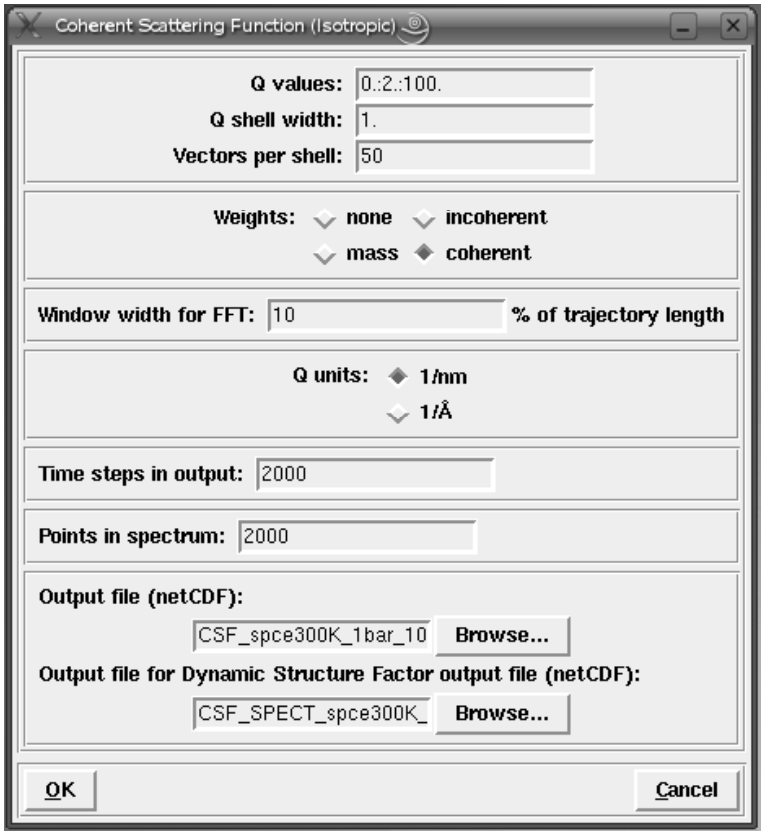

Figure 2.16: Coherent Scattering Function (Isotropic)

the moduli of the required q-vectors through the text field named Q **values:** using the form $q_{min} : \Delta q : q_{max}$. In this way the intermediate scattering function will be calcultad for q_m values defined as $q_m = q_{min} + m \cdot \Delta q$ with m running from 0 to $N_q - 1$ and $q_{max} = (N_q - 1) \cdot \Delta q$.

Through the text field **Vectors per shell**, the user can enter the number

of q-vectors (N_q)that have to be used in the average of Eq. 2.10. The expected value is un integer.

The required tolerance on the q -moduli can be entered via the text-field **Q shell width**. The prgram looks for N_q q-vectors whose moduli deviate from the grid of the selcted values within the given tolerance in order to carry out the average of Eq. 2.10. The **Q shell width** fix the q-resolution.

In the panel **Weights** the user can select how to weight the different contributions to the Intermediate coherent scattering function coming from all atoms of the system (see Section 2.2).

The panel **window width for FFT: percento of trajectory lenght** allows one to select the width of the Gaussian function to be used in the smoothing procedure for the calculation of the Fourier spectrum -see Section 4.1. The width is defined with respect to the lenght of the trajectory.

The units of the q-vectors can be selcted in the panel **q units**. The number N_t of points in the output file containing the intermediate coherent scattering functions and the number of frequency points in the output file containing their fourier spectra can be entered in the panels **time steps in output** and **Points in spectrum**, respectively.

The output files are two netCDF files containing the Coherent intermediate scattering functions as a function of the time (in ps) for all q-values, and their Fourier spectra as a function of the frequency (in THz), respectively. Since $\mathcal{F}_{coh}(q_m, k \cdot \Delta t)$ is symmetric in time, it is stored only for the positive time axis and its Fourier spectrum is stored only for values on the positive frequency axis. The default names of the two output files contain the strings **CSF_** and **CSF_SPECT** respectively, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of nMoldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

Anisotropic Through this option, the average of Eq. 2.10 will be carried out taking the q-vectors lying on a given plane or along a given direction whose moduli deviate within the prescribed tolerance from the required values. Pressing this button the window **Coherent Scattering Function (Anisotropic)** pops-up (see Fig. 2.17). The input parameters to be specified in order to carry out the computation the coherent intermediate scattering function and its Fourier spectrum are the same as for the **Isotropic** option. In addition, here one has to specify the plane or the direction on which the q vectors have to be taken. This can be done via the text-field **Q direction or plane** entering either a unit vector in the form (x, y, z) or two vectors in the form $x, y, z; x', y', z'$ defining a plane.

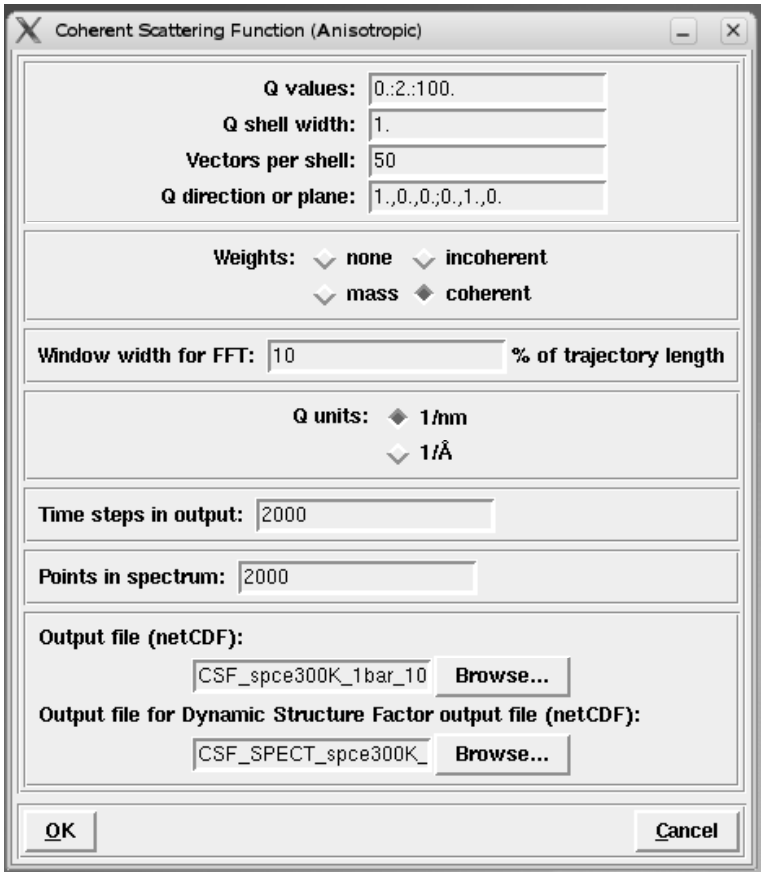

Figure 2.17: Coherent Scattering Function (Anisotropic)

Explicit List Through this option, the average of Eq. 2.10 will be carried out taking the q-vectors whose moduli deviate within the prescribed tolerance from the required values and lying along explicitly listed directions . Pressing this button the window **Coherent Scattering Function (Explicit List)** pops-up (see Fig. 2.18). The input parameters to be specified in order to carry out the computation the coherent intermediate scattering function and its Fourier spectrum are the same as for the **Isotropic** option. In addition, here one has to specify the directions along which the q vectors have to be taken. This can be done via the text-field **Q List** entering a list of unit vectors in the form $x, y, z; x', y', z'; ....$

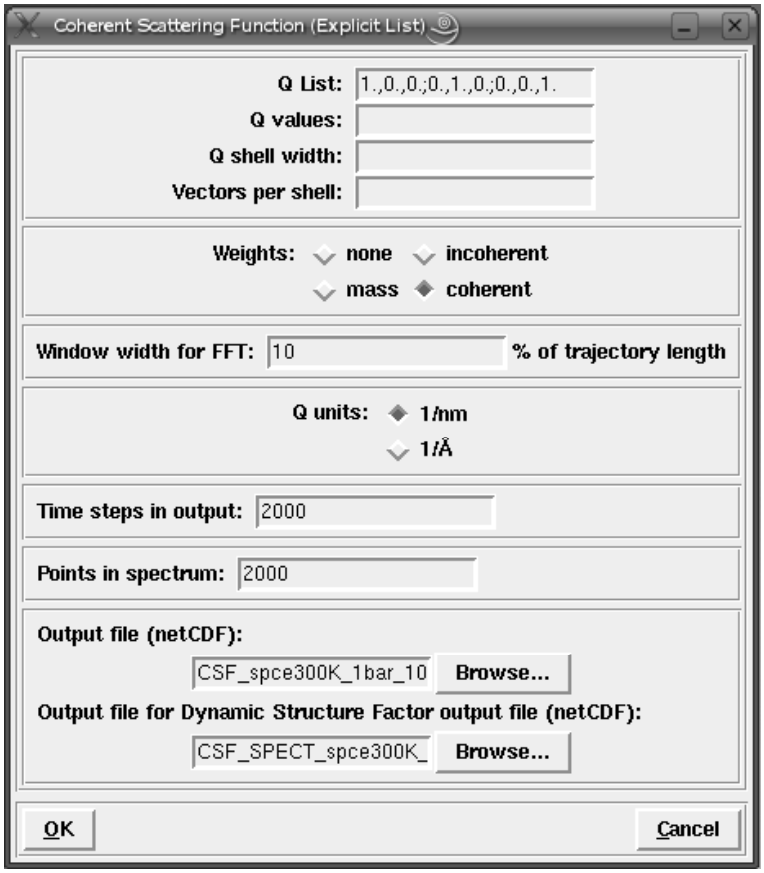

Figure 2.18: Coherent Scattering Function (Explicit List)

Coherent Scattering AR analysis

In the framework of the Autoregressive model, *n*Moldyn allows the intermediate coherent scattering function, its Fourier spectrum (the coherent dynamical structure factor) and its memory function to be computed on a rectangular grid of equidistantly spaced points along the time- and the q-axis, repectively. The user is referred to Section 3.9 for more theoretical details. The dynamical variable of the correlation function under consideration, $\sum_{\alpha=1}^{N} b_{\alpha,coh} \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(n\Delta t)]$, is considered as a discrete "signal", which is modeled by an autoregressive stochastic process of order P. For each q-values the program calculates the set of the relevant P complex coefficients $\{a_n\}$ of the stochastc process, averaging aver all atoms of the system and over all cartesian components. The correlation functions and their Fourier spectra are then computed according to the algorithm described in Section 4.4. Starting from the discretized memory

function equation, which relates the time evolution of the correlation function to its memory function, and using the correlation function calculated by the AR model, the program computes for each q-value the discretized memory function(see Section 4.4). The program performs the above calculations isotropically. Pressing the button **Coherent Scattering AR analysis**, the relevant window pops up (see Fig.2.19).

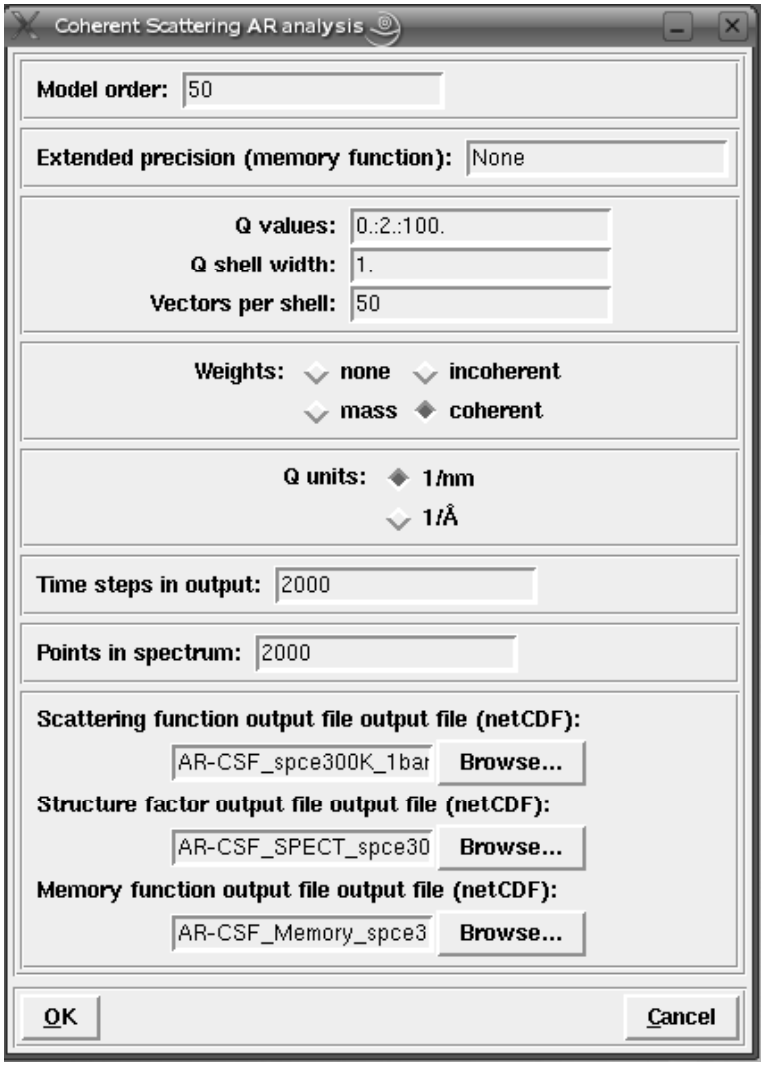

Figure 2.19: Coherent Scattering AR analysis

Here, the user can enter the order (= poles number) of the Autoregressive model in the field **Model Order** (see Section 4.4. *A priori* the autocorrelation function and its power spectrum can be approximated to almost arbitrary precision by increasing the order of the autoregressive model. In practice it has been proven that reliably computation can be carried out up to P of the order of 1000 poles. Since the program performs the calculations in high precision by default, type *None* in the text field **Extended precision (memory function)**. The user can enter the moduli of the required q-values through the text field named Q **values:** using the form $q_{min} : \Delta q : q_{max}$. In this way all of the functions will be calcultad for q_m values defined as $q_m = q_{min} + m \cdot \Delta q$ with m running from 0 to $N_q - 1$ and $q_{max} = (N_q - 1) \cdot \Delta q$. Through the text field **Vectors per shell**, the user can enter the number of q-vectors (N_q) oriented isotropically that have to be used in the average. The expected value is un integer.

The required tolerance on the q-moduli can be entered via the text-field **Q shell width**. The prgram looks for N_q q-vectors whose moduli deviate from the grid of the selcted values within the given tolerance in order to carry out the average. The **Q shell width** fix the q-resolution.

In the panel **Weights** the user can select how to weight the different contributions coming from all atoms of the system (see Section 2.2). The weights are normalized to one.

The units of the q-vectors can be selcted in the panel **q units**. The number N_t of points in the output file containing the intermediate coherent scattering functions and the number of frequency points in the output file containing their fourier spectra can be entered in the panels **time steps in output** and **Points in spectrum**, respectively.

The output files are three netCDF files containing the Coherent intermediate scattering functions as a function of the time (in ps) for all q-values, their Fourier spectra as a function of the frequency (in THz), and their memory functions (in ps). Since $\mathcal{F}_{coh}(q_m, k \cdot \Delta t)$ is symmetric in time, it is stored only for the positive time axis and its Fourier spectrum is stored only for values on the positive frequency axis. The default names of the output files contain the strings **AR-CSF**, **AR-CSF_SPECT**, and **AR-CSF_Memory**, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language (see Fig. 2.20). These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of nMoldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

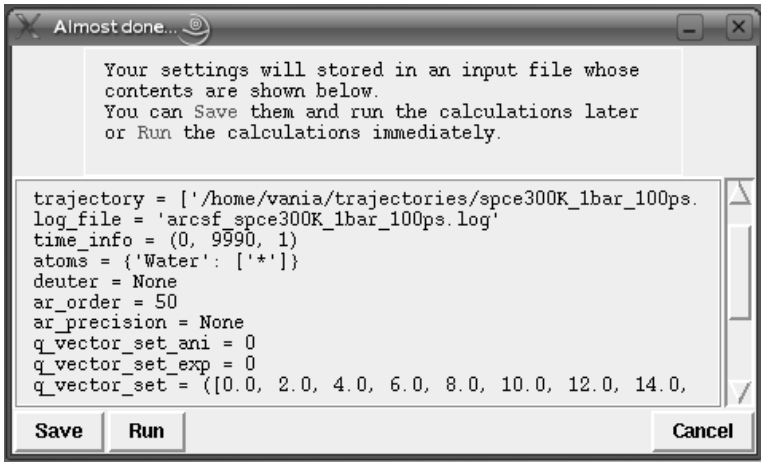

Figure 2.20: Coherent Scattering AR analysis. Python script

Incoherent Scattering Function (Gaussian Approx.)

*n*Moldyn allows one to compute the incoherent intermediate scattering function in the *Gaussian approximation* (see Eq. (3.35)):

$$\mathcal{F}^g_{inc}(q_m, k\cdot\Delta t) \doteq \sum_\alpha w_\alpha F^g_{inc,\alpha}(q_m, k\cdot\Delta t), \qquad k = 0\ldots N_t-1,\ m = 0\ldots N_q-1. \tag{2.15}$$

N_t is the total number of time steps in the coordinate time series and N_q is a user-defined number of q-shells. The (q,t)-grid is the same as for the calculation of the intermediate incoherent scatering function (see Paragraph 2.2.5). The w_α are user-defined weights which are normalized to one, $\sum_\alpha w_\alpha = 1$, such that $\mathcal{F}^g_{inc}(q,0) = 1$. The atomic intermediate scattering functions are given by

$$F^g_{inc,\alpha}(q_m, k\cdot\Delta t) = \exp\left[-\frac{(q_m)^2}{6}\Delta^2_\alpha(k\cdot\Delta t)\right] \qquad \text{isotropic system,} \tag{2.16}$$

$$F^g_{inc,\alpha}(q_m, k\cdot\Delta t) = \exp\left[-\frac{(q_m)^2}{2}\Delta^2_\alpha(k\cdot\Delta t;\mathbf{n})\right] \qquad \text{non-isotropic system.} \tag{2.17}$$

The quantities $\Delta^2_\alpha(t)$ and $\Delta^2_\alpha(t;\mathbf{n})$ are the mean-square displacements, defined in Equations (3.31) and (3.32), respectively. They are computed by using the algorithm described in Section 4.2. *n*Moldyn corrects the atomic input trajectories for jumps due to periodic boundary conditions. It should be noted that the computation of the intermediate scattering function in the Gaussian approximation is much 'cheaper' than the computation of the full intermediate scattering function, $\mathcal{F}_{inc}(q,t)$, since no averaging over different q-vectors needs to be performed. It is sufficient to compute a single mean-square displacement per atom. Pressing the button **Incoherent Scattering Function**

(Gaussian Approx.) in the graphical interface the window *Incoherent Scattering Function (Gaussian Approx.)* pops-up. The user can enter the moduli

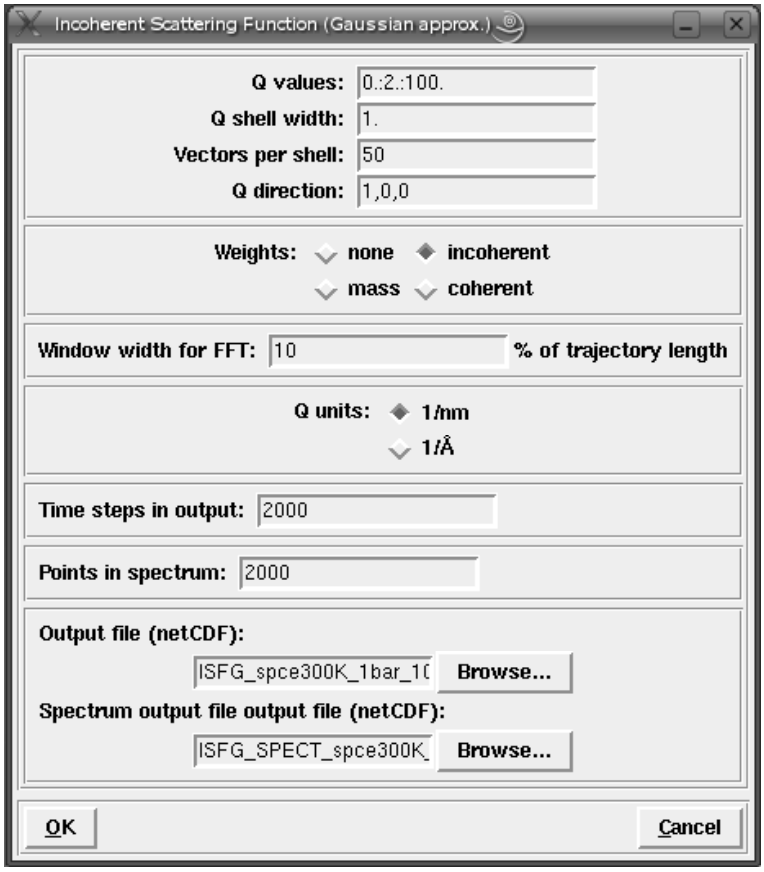

Figure 2.21: Incoherent Scattering Function (Gaussian Approx.)

of the required q-vectors through the text field named Q **values:** using the form $q_{min} : \Delta q : q_{max}$. In this way the intermediate scattering function will be calcultad for q_m values defined as $q_m = q_{min} + m \cdot \Delta q$ with m running from 0 to $N_q - 1$ and $q_max = (N_q - 1) \cdot \Delta q$.

*N*Moldyn can calculate for each atom the mean square displacement with respect to a given axis by entering in the **Projection Vector** field the components of the corresponding unit vector in the form *x,y,z*. Otherwise writing *None*, the MSD will be calculated by averaging over all the directions.

Attention, the values **Vectors per shell** and **Q shell width** are not used to carry out the computation of the intermediate incoherent scattering function in the Gaussian aproximation, since no averaging over different q-vectors needs to be performed. The relative fields must be ignored.

In the panel **Weights** the user can select how to weight the different contributions to the Intermediate coherent scattering function coming from all atoms of the system (see Section 2.2).

The panel **window width for FFT: percento of trajectory lenght** allows one to select the width of the Gaussian function to be used in the smoothing procedure for the calculation of the Fourier spectrum of the intermediate scattering function -see Section 4.1. The width is defined with respect to the lenght of the trajectory.

The units of the q-vectors can be selcted in the panel **q units**. The number N_t of points in the output file containing the intermediate incoherent scattering function and the number of frequency points in the output file containing its fourier spectrum can be entered in the panels **time steps in output** and **Points in spectrum**, respectively.

The output files are two netCDF files containing the incoherent intermediate scattering functions as a function of the time (in ps) for all q-values, and their Fourier spectra as a function of the frequency (in THz), respectively. Since $\mathcal{F}^{\}}_{inc}(q_m, k \cdot \Delta t)$ is symmetric in time, it is stored only for the positive time axis and its Fourier spectrum is stored only for values on the positive frequency axis. The default names of the two output files contain the strings **ISFG** and **ISFG_SPECT** respectively, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately throgh the button **Run**. Pressing the button **cancel** closes the window without saving settings.

Incoherent Scattering Function

Option **Incoherent scattering Function** in the drop-down menu **dynamics** allows one to compute the incoherent intermediate scattering function on a rectangular grid of equidistantly spaced points along the time- and the q-axis, repectively:

$$\mathcal{F}_{inc}(q_m, k \cdot \Delta t) \doteq \sum_{\alpha} w_\alpha F_{inc,\alpha}(q_m, k \cdot \Delta t), \tag{2.18}$$

$$F_{inc,\alpha}(q_m, k \cdot \Delta t) = \overline{\langle \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(0)] \exp[i\mathbf{q} \cdot \mathbf{R}_\alpha(t)] \rangle}^{q}. \tag{2.19}$$

The indices k and m run from 0 to $N_t - 1$ and from 0 to $N_q - 1$, respectively. N_t is the total number of time steps in the coordinate time series and N_q is a user-defined number of q-shells. The values for q_m are defined as $q_m = q_{min} + m \cdot \Delta q$. The program corrects the atomic input trajectories for jumps due to periodic boundary conditions. The symbol $\overline{\ldots}^q$ in (2.19) denotes an average over q-vectors having the same modulus q_m. For each q-value and each atom an average over intermediate scattering functions $F_{inc,\alpha}(\mathbf{q}_i, k \cdot \Delta t)$ for a fixed number of

vectors $\mathbf{q}_i$ is performed. The $\mathbf{q}_i$ vectors can be isotropically distributed in the space, in a plane, or along a direction. The correlation functions $F_{inc,\alpha}(\mathbf{q}_i, k \cdot \Delta t)$ are computed by using the FCA-algorithm described in Section 4.1. The quantities w_α are user-defined weights which are normalized to one, $\sum_\alpha w_\alpha = 1$, such that $\mathcal{F}_{inc}(q, 0) = 1$.

Although the efficient FCA technique is used to compute the atomic time correlation functions, the program may consume a considerable amount of CPU-time since the number of time correlation functions to be computed equals the number of atoms times the total number of q-vectors.

The above calculations may be run via the graphical interface. Pressing the button textbfIncoherent Scattering Function brings up a drop-down menu from which the user can choose whether to calculate $\mathcal{F}_{inc}(q_m, k \cdot \Delta t)$ isotropically ($\rightarrow$ **Isotropic**), or on a given plane ($\rightarrow$ **Anisotropic**), or along a given direction ($\rightarrow$ **Explicit List**):

Isotropic Through this option, the average of Eq. 2.19 will be carried out taking isotropically the q-vectors whose moduli deviate within the prescribed tolerance from the required values. Pressing this button the window **Incoherent Scattering Function (Anisotropic)** pops-up

The input quantities to be entered are the same as for the isotropic calculation of the Coherent scattering.

Here, the output files are two netCDF files containing the Incoherent intermediate scattering functions as a function of the time (in ps) for all q-values, and their Fourier spectra as a function of the frequency (in THz), respectively. Since $\mathcal{F}_{inc}(q_m, k \cdot \Delta t)$ is symmetric in time, it is stored only for the positive time axis and its Fourier spectrum is stored only for values on the positive frequency axis. The default names of the two output files contain the strings **ISF** and **ISF_SPECT** respectively, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of nMoldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

Anisotropic Through this option, the average of Eq. 2.19 will be carried out taking the q-vectors lying on a given plane or along a given direction whose moduli deviate within the prescribed tolerance from the required values. Pressing this button the window **Incoherent Scattering Function (Anisotropic)** pops-up The input parameters to be specified in order to carry out the computation the coherent intermediate scattering function and its Fourier spectrum are the same as for the **Isotropic** option. In addition, here one has to specify the plane or the direction on which the q vectors have to be taken. This can be done via the text-field **Q direction or plane** entering either a unit vector in

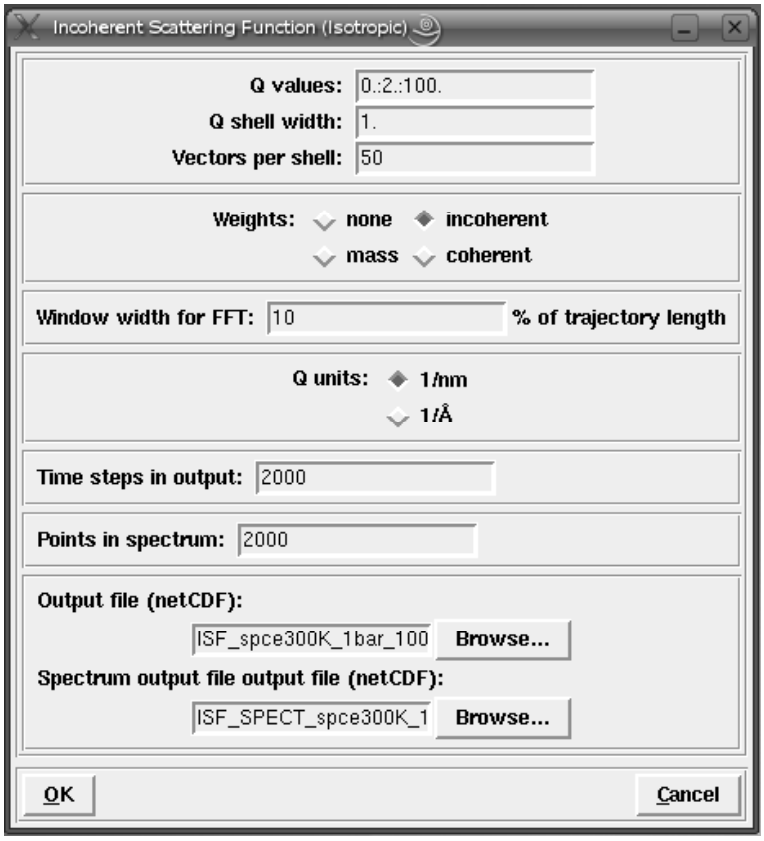

Figure 2.22: Incoherent Scattering Function (Isotropic)

the form (x, y, z) or two vectors in the form $x, y, z; x', y', z'$ defining a plane.

Explicit List Through this option, the average of Eq. 2.19 will be carried out taking the q-vectors whose moduli deviate within the prescribed tolerance from the required values and lying along explicitly listed directions . Pressing this button the window **Incoherent Scattering Function (Explicit List)** pops-up Here the input parameters to be specified in order to carry out the computation the incoherent intermediate scattering function and its Fourier spectrum are the same as for the **Isotropic** option. In addition, here one has to specify the directions along which the q vectors have to be taken. This can be done via the text-field **Q List** entering a list of unit vectors in the form $x, y, z; x', y', z'; ....$

Incoherent Scattering AR analysis

In the framework of the Autoregressive model *n*Moldyn allows the intermediate coherent scattering function, its Fourier spectrum (the incoherent dynamical structure factor) and its memory function to be computed on a rectangular grid of equidistantly spaced points along the time- and the q-axis, repectively. The user is referred to Section 3.9 for more theoretical details. The dynamical variable of the correlation function under consideration, $\sum_{\alpha=1}^{N} b_{\alpha,inc} \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(n\Delta t)]$, is considered as a discrete "signal", which is modeled by an autoregressive stochastic process of order P. For each q-values the program calculates a set of P complex coefficients a_n for the AR model averaging aver all atoms of the system and over all cartesian components. The correlation functions and their Fourier spectra are then computed according to the algorithm described in Section 4.4. Starting from the discretized memory function equation, whitch relates the time evolution of the correlation function to its memory function (see Section 4.4), and using the correlation function calculated by the AR model, the program computes for each q-value the discretized memory function. The program performs the above calculations isotropically. Pressing the button **Incoherent Scattering AR analysis**, the relevant window pops up (see Fig.2.23).

Here, the user can enter the order (= poles number) of the Autoregressive model in the field **Model Order** (see Section 4.4. *A priori* the autocorrelation function and its power spectrum can be approximated to almost arbitrary precision by increasing the order of the autoregressive model. In practice it has been proven that reliably computation can be carried out up to P of the order of 1000 poles. Since the program performs the calculations in high precision by default, type *None* in the text field **Extended precision (memory function)**. The user can enter the moduli of the required q-values through the text field named Q **values:** using the form $q_{min} : \Delta q : q_{max}$. In this way all of the functions will be calcultad for q_m values defined as $q_m = q_{min} + m \cdot \Delta q$ with m running from 0 to $N_q - 1$ and $q_max = (N_q - 1) \cdot \Delta q$. Through the text field **Vectors per shell**, the user can enter the number of q-vectors (N_q) oriented isotropically that have to be used in the average. The expected value is un integer.

The required tolerance on the q-moduli can be entered via the text-field **Q shell width**. The program looks for N_q q-vectors isotropically directed whose moduli deviate from the grid of the selcted values within the given tolerance in order to carry out the average. The **Q shell width** fix the q-resolution.

In the panel **Weights** the user can select how to weight the different contributions coming from all atoms of the system (see Section 2.2). The weights are normalized to one.

The units of the q-vectors can be selcted in the panel **q units**. The number N_t of points in the output file containing the intermediate coherent scattering functions and the number of frequency points in the output file containing their fourier spectra can be entered in the panels **time steps in output** and **Points in spectrum**, respectively.

The output files are three netCDF files containing the Incoherent interme-

diate scattering functions as a function of the time (in ps) for all q-values, their Fourier spectra as a function of the frequency (in THz), and their memory functions (in ps). Since $\mathcal{F}_{inc}(q_m, k \cdot \Delta t)$ is symmetric in time, it is stored only for the positive time axis and its Fourier spectrum is stored only for values on the positive frequency axis. The default names of the output files contain the strings **AR-ISF**, **AR-ISF_SPECT**, and **AR-ISF_Memory**, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately through the button **Run**. Pressing the button **cancel** closes the window without saving settings.

EISF

*n*Moldyn allows one to compute the elastic incoherent structure factor (see Section 3.3) on a grid of equidistantly spaced points along the q-axis:

$$EISF(q_m) \doteq \sum_\alpha w_\alpha EISF_\alpha(q_m), \qquad m = 0 \ldots N_q - 1. \tag{2.20}$$

N_q is a user-defined number of q-shells. The values for q_m are defined as $q_m = q_{min} + m \cdot \Delta q$. The atomic EISFs are computed as

$$EISF_\alpha(q_m) = \overline{\langle |\exp[i\mathbf{q} \cdot \mathbf{R}_\alpha]|^2 \rangle}^q. \tag{2.21}$$

Here the symbol $\overline{\ldots}^q$ denotes an average over the q-vectors having the same modulus q_m. The program corrects the atomic input trajectories for jumps due to periodic boundary conditions. The quantities w_α are user-defined weights which are normalized to one, $\sum_\alpha w_\alpha = 1$, such that $EISF(0) = 1$. Pressing the button **EISF** the corresponding graphical interface pops-up (see Fig. 2.24).

The user can enter the moduli of the required q-vectors through the text field named Q **values:** using the form $q_{min} : \Delta q : q_{max}$. In this way the intermediate scattering function will be calcultad for q_m values defined as $q_m = q_{min} + m \cdot \Delta q$ with m running from 0 to $N_q - 1$ and $q_max = (N_q - 1) \cdot \Delta q$.

Through the text field **Vectors per shell**, the user can enter the number of q-vectors (N_q) that have to be used in the average of Eq. 2.21. The expected value is un integer.

The required tolerance on the q-moduli can be entered via the text-field **Q shell width**. The program looks for N_q q-vectors whose moduli deviate from the grid of the selcted values within the given tolerance in order to carry out the average of Eq. 2.21. The **Q shell width** fix the q-resolution.

In the panel **Weights** the user can select how to weight the different contributions to the Intermediate coherent scattering function coming from all atoms of the system (see Section 2.2).

The units of the q-vectors can be selcted in the panel **q units**.

Here, the output files is a netCDF files containing the EISF as a function of q in the selcted units The default names of the output file contains the strings **EISF**, followed by the name of the trajectory. The user can then change the names and/or the destination directory through the **Browse** buttons.

Pressing the **OK** button switches to the *Almost Done* window containing all settings in python language. These settings can be saved in a python script file via the **Save** button and then run later using the command line interface of *n*Moldyn or run immediately throgh the button **Run**. Pressing the button **cancel** closes the window without saving settings.

2.2.6 Output/Input Inspection: View

Pressing the button **View** brings up a drop-down menu from which it is possible to choose the following options:

- Trajectory information
- View variables
- Animation ...
- Display
- Display 3D
- Reciprocal basis
- Direct basis

In this section we give a description of the above options.

Trajectory information

Pressing this button one can visualize the trajectory information:

- name of the trajectory file and location directory
- simulated system: molecules number, atoms number and time steps.
- creation date

Press the **Close** button to come back to the main start-up window of *n*Moldyn.

View variables

Pressing the button **View variables** a drop-down menu, from which one can select the variable to be inspected, brings up. The option **termodynamics** shows the evolution of the temperature (in Kelvin) during the simulation, while the option **energy** shows the evolution of the kinetic energy ($kJ \cdot mol^{-1}$).

Animation

Animation of the imported trajectoty can be displayed through this option. Pressing button **Animations** a window *trajectory animation* pops-up. From this window the user can enter the first and the last step of the trajectory to be precessed in the text fields **first step** and **last step**, respectively, and the number of time steps to be skipped in the text field **skip**.

Display

This option allows the output functions to be displayed in a 2D plot. For the functions calculated over a (q, t) gride the plots are done kiping fixed one of the two variables.

Display 3D

The output functions calculated aver a (q, t) gride can be displayed in a 3D plot through option **Display3D**.

Direct basis

Pressing this button a window containing the basis vectors $\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}$ which span the MD cell pops-up. Any position vector in the MD call can be written as $\mathbf{R} = x'\mathbf{b_1} + y'\mathbf{b_2} + z'\mathbf{b_3}$, with x', y', z' having values between 0 and 1.

Reciprocal basis

Pressing this button a window containing the dual basis vectors $\mathbf{b^1}, \mathbf{b^2}, \mathbf{b^3}$ pops-up. They are defined by the relation $\mathbf{b}_i\mathbf{b}_j = \delta_i^j$, where $\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}$ are the direct basis vectors.

2.2.7 Help

Pressing the button **Help** brings up a drop-down menu from which it is possible to choose the option **About nMoldyn**. Pressing this button a window containing the e-mail addresses of all of the people which have contributed to the developpement of the code: Gerald Kneller, author of the original fortran version of nMoldyn (release 1.0) Tomasz Róg, Krzysztof Murzyn, Konrad Hinsen authors of the python version of nMoldyn (release 2.1) Paolo Calligari author of a revised version of nMoldyn based on the release 2.1

2.3 Command-Line Interface: pMoldyn

In some situations a graphical interface cannot be used for technical reasons (e.g., text-mode connection to remote machines) or it is not the most convenient solution. This occurs tipically when one need to perform a large number of similar calculations or when high flexibility is required in the selection of

the atoms and groups that are to be used for a given calculation. For these situations, nMoldyn provides a command-line interface that reads all input information from a single specification file. The specification files are Python scripts. In the following we show as an example, the specification file for a simple mean-square displacement calculated over all of the atoms of a protein:

```
from MMTK import *
trajectory = [‘‘lysozyme.nc’’]
output_files = [‘‘msd’’:  ‘‘msd.plot’’]
title = ‘‘Mean Square Displacement’’
time_info = (0, None, 1)
weights = ‘‘mass’’
atoms = (’Protein.0’:  [’*’])
```

The command line interface provides high flexibility for advanced users. For example, if the standard atom selections offered by the graphical interface are insufficients, it is possible to customize the atom selection by few lines of Python in the nMoldyn specification file. The following file specifies the calculation of the mean square displacement for only the sidechains of residues 5 to 25:

```
from MMTK import *
trajectory = [‘‘lysozyme.nc’’]
output_files = [‘‘msd’’:  ‘‘msd.plot’’]
title = ‘‘Mean Square Displacement’’
time_info = (0, None, 1)
weights = ‘‘mass’’
def atoms_code(trajectory):
# retrieve all proteins
from MMTK.Proteins import Protein
proteins = trajectory.universe.\
objectList(Protein)
# pick the lysozyme
lysozyme = proteins[0]
# pick a part of the first chain
subchain = lysozyme[0][4:25]
# select the sidechains
return subchain.sidechains() [1]
```

We recall that it is possible to create an input file via the graphical user interface as well. Such input file provides a convenient starting point for customization.

To run a calculation via the command-line interface type

```
pMoldyn
```

without input arguments in order to get usage instructions.

[1] the two examples of specification file shown above are taken from Ref [39]

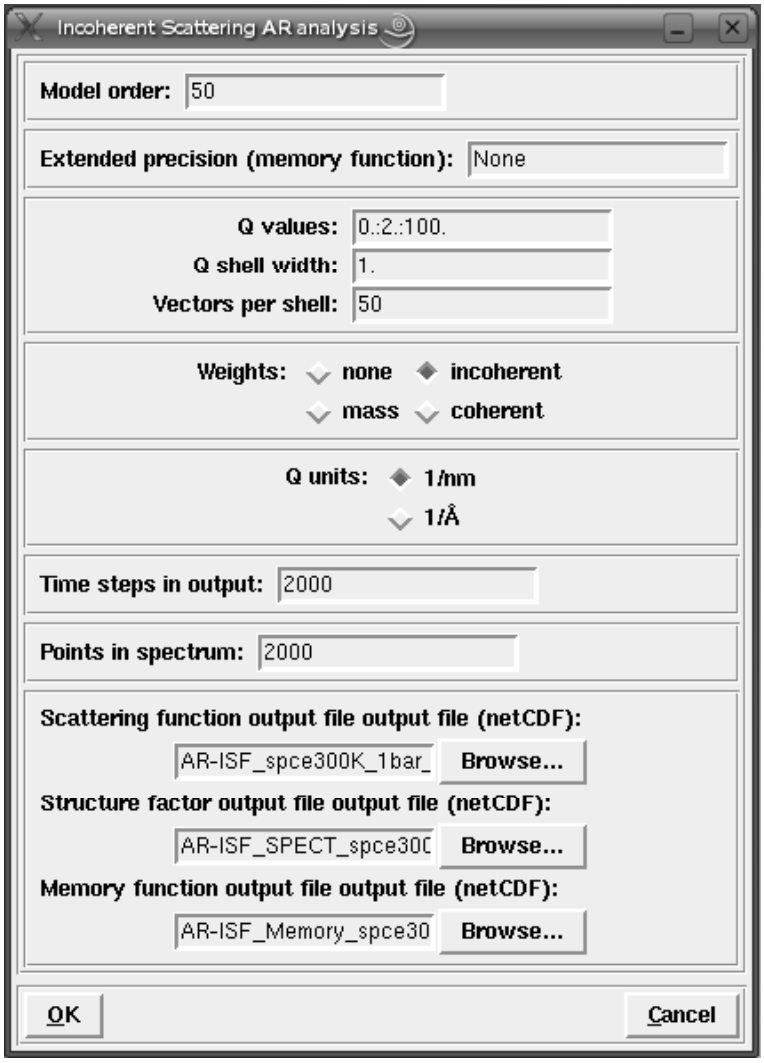

Figure 2.23: Coherent Scattering AR analysis

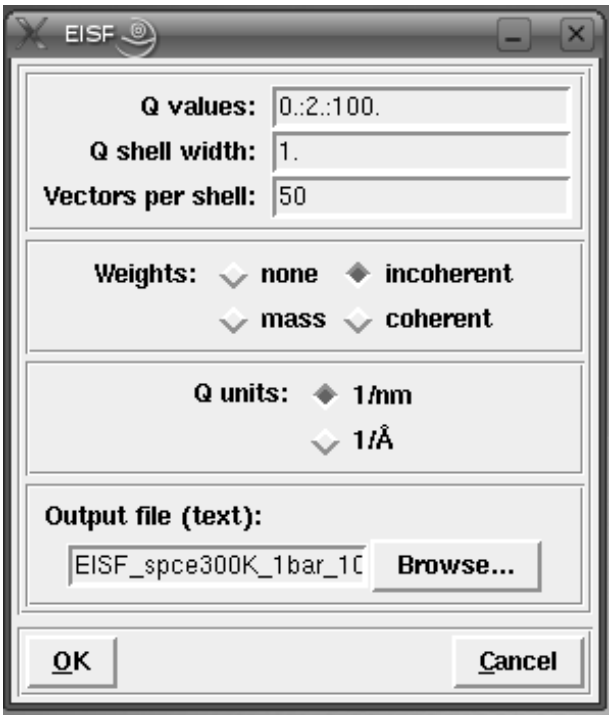

Figure 2.24: EISF

Chapter 3

Theoretical background

3.1 Dynamic structure factor

The quantity of interest in neutron scattering experiments with thermal neutrons is the *dynamic structure factor*, $\mathcal{S}(\mathbf{q},\omega)$, which is closely related to the double differential cross-section [1], $d^2\sigma/d\Omega dE$. The double differential cross section is defined as the number of neutrons which are scattered per unit time into the solid angle interval $[\Omega, \Omega + d\Omega]$ and into the energy interval $[E, E + dE]$. It is normalized to $d\Omega$, dE, and the flux of the incoming neutrons,

$$\frac{d^2\sigma}{d\Omega dE} = N \cdot \frac{k}{k_0}\mathcal{S}(\mathbf{q},\omega). \tag{3.1}$$

Here n is the number of atoms, and $k \equiv |\mathbf{k}|$ and $k_0 \equiv |\mathbf{k}_0|$ are the wave numbers of scattered and incident neutrons, respectively. They are related to the corresponding neutron energies by $E = \hbar^2 k^2/2m$ and $E_0 = \hbar^2 k_0^2/2m$, where m is the neutron mass. The arguments of the dynamic structure factor, $\mathbf{q}$ and ω, are the momentum and energy transfer in units of $\hbar$, respectively:

$$\mathbf{q} = \frac{\mathbf{k}_0 - \mathbf{k}}{\hbar}, \tag{3.2}$$

$$\omega = \frac{E_0 - E}{\hbar}. \tag{3.3}$$

The modulus of the momentum transfer can be expressed in the scattering angle θ, the energy transfer, and the energy of the incident neutrons:

$$q = \sqrt{2 - \frac{\hbar\omega}{E_0} - 2\cos\theta\sqrt{2 - \frac{\hbar\omega}{E_0}}}. \tag{3.4}$$

The dynamic structure factor contains information about the structure and dynamics of the scattering system [21]. It can be written as

$$\mathcal{S}(\mathbf{q},\omega) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} dt\ \exp[-i\omega t]\mathcal{F}(\mathbf{q},t). \tag{3.5}$$

$\mathcal{F}(\mathbf{q},t)$ is called the *intermediate scattering function* and is defined as

$$\mathcal{F}(\mathbf{q},t) = \sum_{\alpha,\beta} \Gamma_{\alpha\beta} \langle \exp[-i\mathbf{q}\cdot\hat{\mathbf{R}}_\alpha(0)] \exp[i\mathbf{q}\cdot\hat{\mathbf{R}}_\beta(t)]\rangle, \tag{3.6}$$

$$\Gamma_{\alpha\beta} = \frac{1}{N}\left[\overline{b_\alpha}\,\overline{b_\beta} + \delta_{\alpha\beta}(\overline{b_\alpha^2} - \overline{b_\alpha}^2)\right]. \tag{3.7}$$

The operators $\hat{\mathbf{R}}_\alpha(t)$ in Eq. (3.6) are the position operators of the nuclei in the sample. The brackets $\langle \ldots \rangle$ denote a quantum thermal average and the time dependence of the position operators is defined by the Heisenberg picture. The quantities b_α are the scattering lengths of the nuclei which depend on the isotope and the relative orientation of the spin of the neutron and the spin of the scattering nucleus. If the spins of the nuclei and the neutron are not prepared in a special orientation one can assume a random relative orientation and that spin and position of the nuclei are uncorrelated. The symbol $\overline{\ldots}$ appearing in $\Gamma_{\alpha\beta}$ denotes an average over isotopes and relative spin orientations of neutron and nucleus.

Usually one splits the intermediate scattering function and the dynamic structure factor into their *coherent* and *incoherent* parts which describe collective and single particle motions, respectively. Defining

$$b_{\alpha,coh} \doteq \overline{b_\alpha}, \tag{3.8}$$

$$b_{\alpha,inc} \doteq \sqrt{\overline{b_\alpha^2} - \overline{b_\alpha}^2}, \tag{3.9}$$

the coherent and incoherent intermediate scattering functions can be cast in the form

$$\mathcal{F}_{coh}(\mathbf{q},t) = \frac{1}{N}\sum_{\alpha,\beta} b_{\alpha,coh}\, b_{\beta,coh} \langle \exp[-i\mathbf{q}\cdot\hat{\mathbf{R}}_\alpha(0)] \exp[i\mathbf{q}\cdot\hat{\mathbf{R}}_\beta(t)]\rangle, \tag{3.10}$$

$$\mathcal{F}_{inc}(\mathbf{q},t) = \frac{1}{N}\sum_{\alpha} b_{\alpha,inc}^2 \langle \exp[-i\mathbf{q}\cdot\hat{\mathbf{R}}_\alpha(0)] \exp[i\mathbf{q}\cdot\hat{\mathbf{R}}_\alpha(t)]\rangle. \tag{3.11}$$

The corresponding dynamic structure factors are obtained by performing the Fourier transformation defined in Eq. (3.5).

An important quantity describing *structural* properties of liquids is the *static structure factor*, which is defined as

$$\mathcal{S}(\mathbf{q}) \doteq \int_{-\infty}^{+\infty} d\omega\, \mathcal{S}_{coh}(q,\omega) = \mathcal{F}_{coh}(\mathbf{q},0). \tag{3.12}$$

If the sums over α and β in (3.10) are extended over subsets of respectively equivalent atoms, one obtains the *partial static structure factors*.

3.2 Classical framework

In the classical framework the intermediate scattering functions are interpreted as classical time correlation functions. The position operators are replaced by

time-dependent vector functions and quantum thermal averages are replaced by classical *ensemble averages*. It is well known that this procedure leads to a loss of the universal detailed balance relation,

$$\mathcal{S}(\mathbf{q},\omega) = \exp[\beta\hbar\omega]\mathcal{S}(-\mathbf{q},-\omega), \tag{3.13}$$

and also to a loss of all odd moments

$$\langle\omega^{2n+1}\rangle \doteq \int_{-\infty}^{+\infty} d\omega\, \omega^{2n+1}\mathcal{S}(\mathbf{q},\omega), \qquad n = 1, 2, \ldots. \tag{3.14}$$

The odd moments vanish since the classical dynamic structure factor is even in ω, assuming invariance of the scattering process with respect to reflections in space. The first moment is also universal. For an atomic liquid, containing only one sort of atoms, it reads

$$\langle\omega\rangle = \frac{\hbar q^2}{2M}\,, \tag{3.15}$$

where M is the mass of the atoms. Formula (3.15) shows that the first moment is given by the average kinetic energy (in units of $\hbar$) of a particle which receives a momentum transfer $\hbar\mathbf{q}$. Therefore $\langle\omega\rangle$ is called the *recoil moment*. A number of 'recipes' has been suggested to correct classical dynamic structure factors for detailed balance and to describe recoil effects in an approximate way. The most popular one has been suggested by Schofield [22]

$$\mathcal{S}(\mathbf{q},\omega) \approx \exp[\frac{\beta\hbar\omega}{2}]\mathcal{S}_{cl}(\mathbf{q},\omega). \tag{3.16}$$

One can easily verify that the resulting dynamic structure factor fulfills the relation of detailed balance. Formally, the correction (3.16) is correct to first order in $\hbar$. Therefore it cannot be used for large q-values which correspond to large momentum transfers $\hbar q$. This is actually true for all correction methods which have suggested so far. For more details we refer to [3].

3.3 Elastic incoherent structure factor

The elastic incoherent structure factor (EISF) is defined as the limit of the incoherent intermediate scattering function for infinite time,

$$EISF(\mathbf{q}) \doteq \lim_{t\to\infty} \mathcal{F}_{inc}(\mathbf{q},t). \tag{3.17}$$

Using the above definition of the EISF one can decompose the incoherent intermediate scattering function as follows:

$$\mathcal{F}_{inc}(\mathbf{q},t) = EISF(\mathbf{q}) + \mathcal{F}'_{inc}(\mathbf{q},t), \tag{3.18}$$

where $\mathcal{F}'_{inc}(\mathbf{q},t)$ decays to zero for infinite time. Taking now the Fourier transform it follows immediately that

$$\mathcal{S}_{inc}(\mathbf{q},\omega) = EISF(\mathbf{q})\delta(\omega) + \mathcal{S}'_{inc}(\mathbf{q},\omega). \tag{3.19}$$

The EISF appears as the amplitude of the *elastic* line in the neutron scattering spectrum. Elastic scattering is only present for sytems in which the atomic motion is confined in space, as for solids. To understand which information is contained in the EISF we consider for simplicity a system where only one sort of atoms is visible to the neutrons. To a very good approximation this is the case for all systems containing a large amount of hydrogen atoms, as biological systems. Incoherent scattering from hydrogen dominates by far all other contributions. Using the definition of the van Hove self-correlation function $G_s(\mathbf{r}, t)$ [1],

$$b_{inc}^2 G_s(\mathbf{r}, t) \doteq \frac{1}{2\pi^3} \int d^3q \, \exp[-i\mathbf{q} \cdot \mathbf{r}] \mathcal{F}_{inc}(\mathbf{q}, t), \tag{3.20}$$

which can be interpreted as the conditional probability to find a tagged particle at the position $\mathbf{r}$ at time t, given it started at $\mathbf{r} = \mathbf{0}$, one can write:

$$EISF(\mathbf{q}) = b_{inc}^2 \int d^3r \, \exp[i\mathbf{q} \cdot \mathbf{r}] \, G_s(\mathbf{r}, t = \infty). \tag{3.21}$$

The EISF gives the sampling distribution of the points in space in the limit of infinite time. In a real experiment this means times longer than the time which is observable with a given instrument. The EISF vanishes for all systems in which the particles can access an infinite volume since $G_s(\mathbf{r}, t)$ approaches $1/V$ for large times. This is the case for molecules in liquids and gases.

For computational purposes it is convenient to use the following representation of the EISF [8]:

$$EISF(\mathbf{q}) = \frac{1}{N} \sum_{\alpha} b_{\alpha,inc}^2 \langle | \exp[i\mathbf{q} \cdot \mathbf{R}_\alpha] |^2 \rangle. \tag{3.22}$$

This expression is derived from definition (3.17) of the EISF and expression (3.11) for the intermediate scattering function, using that for infinite time the relation

$$\langle \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(0)] \exp[i\mathbf{q} \cdot \mathbf{R}_\alpha(t)] \rangle = \langle | \exp[i\mathbf{q} \cdot \mathbf{R}_\alpha] |^2 \rangle \tag{3.23}$$

holds. In this way the computation of the EISF is reduced to the computation of a static thermal average. We remark at this point that the length of the MD trajectory from which the EISF is computed should be long enough to allow for a representative sampling of the conformational space.

3.4 Velocity correlation functions and Density of States

The velocity autocorrelation function (VACF) of atom α in an atomic or molecular system is usually defined as

$$C_{vv;\alpha\alpha}(t) \doteq \frac{1}{3} \langle \mathbf{v}_\alpha(0) \cdot \mathbf{v}_\alpha(t) \rangle. \tag{3.24}$$

In some cases, e.g. for non-isotropic systems, it is useful to define VACFs along a given axis,

$$C_{vv;\alpha\alpha}(t;\mathbf{n}) \doteq \langle v_\alpha(0;\mathbf{n}) v_\alpha(t;\mathbf{n})\rangle, \tag{3.25}$$

where $v_\alpha(t;\mathbf{n})$ is given by

$$v_\alpha(t;\mathbf{n}) \doteq \mathbf{n}\cdot\mathbf{v}_\alpha(t). \tag{3.26}$$

The vector $\mathbf{n}$ is a unit vector defining a space-fixed axis.

The VACFs of the particles in a many body system can be related to the incoherent dynamic structure factor. It is easy to show that

$$lim_{q\to 0}\frac{\omega^2}{q^2}\mathcal{S}(\mathbf{q},\omega) = G(\omega), \tag{3.27}$$

where $G(\omega)$ is the *density-of-states* (DOS). For an isotropic system it reads

$$G(\omega) = \sum_\alpha b^2_{\alpha,inc}\tilde{C}_{vv;\alpha\alpha}(\omega), \tag{3.28}$$

$$\tilde{C}_{vv;\alpha\alpha}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} dt\, \exp[-i\omega t] C_{vv;\alpha\alpha}(t). \tag{3.29}$$

For non-isotropic systems relation (3.27) holds if the DOS is computed from the atomic velocity autocorrelation functions $C_{vv;\alpha\alpha}(t;\mathbf{n}_q)$, where $\mathbf{n}_q$ is the unit vector in the direction of $\mathbf{q}$.

3.5 Mean-square displacements

Another important quantity describing the dynamics of an atomic or molecular system is the mean-square displacement (MSD) of a particle. Defining

$$\mathbf{d}_\alpha(t) \doteq \mathbf{R}_\alpha(t) - \mathbf{R}_\alpha(0). \tag{3.30}$$

the MSD of particle α is given by

$$\Delta^2_\alpha(t) = \left\langle \mathbf{d}^2_\alpha(t)\right\rangle. \tag{3.31}$$

As for the VACF one can introduce a mean-square displacement with respect to a given axis:

$$\Delta^2_\alpha(t;\mathbf{n}) \doteq \left\langle d^2_\alpha(t;\mathbf{n})\right\rangle \tag{3.32}$$

with

$$d_\alpha(t;\mathbf{n}) \doteq \mathbf{n}\cdot\mathbf{d}_\alpha(t). \tag{3.33}$$

The calculation of mean-square displacements is the standard way to obtain diffusion coefficients from MD simulations. Assuming Einstein-diffusion in the long time limit one has for isotropic systems

$$D_\alpha = \lim_{t\to\infty}\frac{1}{6t}\Delta^2_\alpha(t). \tag{3.34}$$

The mean-square displacement can be related to the incoherent intermediate scattering function via the cumulant expansion [23, 24]

$$\mathcal{F}_{inc}(\mathbf{q},t) = \frac{1}{N}\sum_{\alpha} b_{\alpha,inc}^2 \exp[-q^2\rho_{\alpha,1}(t) + q^4\rho_{\alpha,2}(t) \mp \ldots]. \tag{3.35}$$

The cumulants $\rho_{\alpha,k}(t)$ are defined as

$$\begin{aligned} \rho_{\alpha,1}(t) &= \frac{1}{2!}\langle d_\alpha^2(t;\mathbf{n}_q)\rangle & (3.36)\\ \rho_{\alpha,2}(t) &= \frac{1}{4!}\left[\langle d_\alpha^4(t;\mathbf{n}_q)\rangle - 3\langle d_\alpha^2(t;\mathbf{n}_q)\rangle^2\right] & (3.37)\\ &\vdots \end{aligned}$$

The vector $\mathbf{n}_q$ is the unit vector in the direction of $\mathbf{q}$. In the Gaussian approximation the above expansion is truncated after the q^2-term. For certain model systems like the ideal gas, the harmonic oscillator, and a particle undergoing Einstein diffusion, this is exact. For these systems the incoherent intermediate scattering function is completely determined by the mean-square displacement.

There exists also a well-known relation between the mean-square displacement and the velocity autocorrelation function. Writing $\mathbf{d}_\alpha(t) = \int_0^t d\tau\, \mathbf{v}_\alpha(\tau)$ in Eq. (3.31) one can show (see e.g. [24]) that

$$\Delta_\alpha^2(t) = 6\int_0^t d\tau\,(t-\tau)C_{vv;\alpha\alpha}(\tau). \tag{3.38}$$

Using now the definition (3.34) of the diffusion coefficient one obtains the relation

$$D_\alpha = \int_0^t d\tau\, C_{vv;\alpha\alpha}(\tau). \tag{3.39}$$

With Eq. (3.29) this can also be written as

$$D_\alpha = \pi\tilde{C}_{vv;\alpha\alpha}(0). \tag{3.40}$$

3.6 Rigid-body motions

To analyze the dynamics of complex molecular systems it is often desirable to consider the overall motion of molecules or molecular subunits. We will call this motion rigid-body motion in the following. Rigid-body motions are fully determined by the dynamics of the centroid, which may be the center-of-mass, and the dynamics of the angular coordinates describing the orientation of the rigid body. The angular coordinates are the appropriate variables to compute angular correlation functions of molecular systems in space and time. In most cases, however, these variables are not directly available from Molecular Dynamics simulations since MD algorithms typically work in cartesian coordinates. Molecules are either treated as flexible, or, if they are treated as rigid, constraints are taken into

account in the framework of cartesian coordinates [25]. In *n*MOLDYN rigid-body trajectories can be extracted from a Molecular Dynamics trajectory by fitting rigid reference structures, defining a (sub)molecule, to the corresponding structure in each time frame of the trajectory. Here 'fit' means the optimal superposition of the structures in a least-squares sense. The corresponding algorithm is described in Section 4.3. It uses *quaternion parameters* [26] for the rotational fit. Quaternions are not only useful for the fitting procedure, but also for the computation of angular correlation functions [27, 28].

3.7 Angular VACF, angular trajectories

Similarly to the translational velocity autocorrelation functions introduced in Section 2.2.4 one can define angular velocity autocorrelation functions to characterize the angular motion of molecules. In general the angular velocity is referred to an orthonormal *body-fixed* coordinate system. Usually this is the principal axis system in which the tensor of inertia is diagonal. Depending on its geometry, a molecule will behave differently with respect to rotational motion about different body-fixed axes. The autocorrelation function for the angular velocity components ω_i' is defined as

$$C_{\omega\omega}(t;i) \doteq \langle \omega_i'(0)\omega_i'(t)\rangle. \tag{3.41}$$

The prime indicates a body fixed coordinate system. The components ω_i' are related to the quaternion parameters describing the orientation of the molecule and their time derivatives [8, 29]:

$$\begin{pmatrix} 0 \\ \omega_x' \\ \omega_y' \\ \omega_z' \end{pmatrix} = 2 \cdot \begin{pmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{pmatrix} \begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix}. \tag{3.42}$$

Here the quaternion parameters describe the rotation of the space-fixed coordinate system into the body-fixed coordinate system. The corresponding rotation matrix is explicitly given in Eq. (4.28).

The components of the angular velocity may be used to define rotation angles describing rotations about the body-fixed axes [8]:

$$\Phi_i(t) = \int_0^t d\tau\, \omega_i'(\tau). \tag{3.43}$$

3.8 Reorientational correlation functions

The molecular reorientational correlation function is defined as the conditional probability to find a molecule with orientation $\mathbf{\Omega}_1$ at time t_1, given it had the orientation $\mathbf{\Omega}_0$ at time t_0. In the following this probability will be denoted by $p(\mathbf{\Omega}_1, t_1|\mathbf{\Omega}_0, t_0)$. Here $\mathbf{\Omega}$ denotes a set of angular coordinates, as Euler angles or

quaternion parameters. The joint probability $p(\mathbf{\Omega}_1, t_1; \mathbf{\Omega}_0, t_0)$ which gives the probablity to find a molecule with orientation $\mathbf{\Omega}_0$ at time t_0 and with orientation $\mathbf{\Omega}_1$ at time t_1, can be expressed as $p(\mathbf{\Omega}_1, t_1; \mathbf{\Omega}_0, t_0) = p(\mathbf{\Omega}_1, t_1|\mathbf{\Omega}_0, t_0) \cdot p(\mathbf{\Omega}_0, t_0)$. Here $p(\mathbf{\Omega}_0, t_0)$ is the probability to find a molecule with orientation $\mathbf{\Omega}_0$ at time t_0. If we consider an isotropic system in thermal equilibrium the reorientational correlation function depends only on the time *difference*, $t = t_1 - t_0$, and the *change* in orientation, $\mathbf{\Omega}$, i.e. $p(\mathbf{\Omega}_1, t_1|\mathbf{\Omega}_0, t_0) = p(\mathbf{\Omega}, t|\mathbf{0}, 0)$. In addition we have $p(\mathbf{\Omega}_0, t_0) = 1/(8\pi^2)$, where $8\pi^2$ is the volume of the angular space.

The reorientational correlation function may now be expanded in Wigner rotation matrices [28] which form a complete set of basis functions in $\mathbf{\Omega}$ [30, 31]:

$$p(\mathbf{\Omega}, t|\mathbf{0}, 0) = \sum_{jmn} \frac{2j+1}{8\pi^2} p^j_{mn}(t) D^{*\,j}_{mn}(\mathbf{\Omega}). \tag{3.44}$$

In the following the coefficients $p^j_{mn}(t)$ are called p-coefficients. Using the orthogonality of the Wigner functions,

$$\int d\Omega\, D^{*\,j}_{mn}(\mathbf{\Omega}) D^{j'}_{m'n'}(\mathbf{\Omega}) = \frac{8\pi^2}{2j+1} \delta_{jj'} \delta_{mm'} \delta_{nn'}, \tag{3.45}$$

Eq. (3.44) can be inverted to give:

$$p^j_{mn}(t) = \int d\Omega\, p(\mathbf{\Omega}, t|\mathbf{0}, 0) D^j_{mn}(\mathbf{\Omega}). \tag{3.46}$$

Writing the reorientational correlation function as

$$p(\mathbf{\Omega}, t|\mathbf{0}, 0) = \frac{1}{N} \sum_\alpha \langle \delta[\mathbf{\Omega} - \mathbf{\Omega}_\alpha(t)] \rangle, \tag{3.47}$$

where $\mathbf{\Omega}_\alpha(t)$ is the orientation of molecule α with respect to its initial orientation and $\langle \ldots \rangle$ is a thermal average, relation (3.46) can be written as

$$p^j_{mn}(t) = \frac{1}{N} \sum_\alpha \langle D^j_{mn}(\mathbf{\Omega}_\alpha(t)) \rangle. \tag{3.48}$$

The p-coefficients can also be expressed as time correlation functions of irreducible tensor components. This is convenient for numerical purposes since time correlation functions of discrete and finite time series can be very efficiently computed by Fast Fourier Transform techniques (see Section 4.1). Consider the general form of the time correlation function

$$\langle T^j_m(t_1) T^{*\,j}_n(t_0) \rangle = \int\int d\Omega_1 d\Omega_0\, p(\mathbf{\Omega}_1, t_1; \mathbf{\Omega}_0, t_0) T^j_m(\mathbf{\Omega}_1) T^{*\,j}_n(\mathbf{\Omega}_0), \tag{3.49}$$

where T^j_m are the components of an *irreducible tensor* [30][31]. From the transformation properties of irreducible tensors it follows that

$$T^j_m(\mathbf{\Omega}_1) = \sum_k D^j_{mk}(\mathbf{\Omega}) T^j_k(\mathbf{\Omega}_0). \tag{3.50}$$

For an isotropic system in thermal equilibrium we may now write

$$p(\mathbf{\Omega}_1, t_1; \mathbf{\Omega}_0, t_0) = p(\mathbf{\Omega}, t|\mathbf{0}, 0) \cdot \frac{1}{8\pi^2}. \tag{3.51}$$

Inserting this in (3.49), performing a change in the integration variables from $(\mathbf{\Omega}_1, \mathbf{\Omega}_0)$ to $(\mathbf{\Omega}, \mathbf{\Omega}_0)$, and using the orthogonality of the Wigner functions one can show that

$$\langle T_m^j(t) T_n^{*\,j}(0)\rangle = p_{mn}^j(t) \cdot \frac{1}{2j+1} \sum_l |\hat{T}_l^j|^2, \tag{3.52}$$

where the components $\hat{T}_l^j$ are referred to a convenient reference frame. In practice only tensors with integer j are relevant. In this case, the well known spherical harmonics [30][31] may be used to define irreducible tensors. They are related to the Wigner functions by

$$Y_m^j(\alpha, \beta) = \sqrt{\frac{2j+1}{4\pi}} D_{m0}^j(\alpha, \beta, \gamma), \tag{3.53}$$

where α, β, γ are Euler angles. Following Rose [32] the Wigner functions can be expressed as complex polynomials in the quaternion parameters:

$$\begin{aligned} D_{mn}^j(\mathbf{q}) = \textstyle\sum_p (-1)^p \frac{[(j+m)!(j-m)!(j+n)!(j-n)!]^{1/2}}{(j+m-p)!(j-n-p)!p!(p+n-m)!} \\ \times (q_0 + iq_3)^{j+m-p} (q_0 - iq_3)^{j-n-p} (q_2 + iq_1)^{p+n-m} (q_2 - iq_1)^p. \end{aligned} \tag{3.54}$$

Here the quaternion parameters describe the rotation of the space-fixed coordinate system into the body-fixed coordinate system. The corresponding rotation matrix is given in Eq. (4.28). According to Eq. (3.53) the spherical harmonics are just special cases of the Wigner functions,

$$\begin{aligned} Y_m^j(\mathbf{q}) = \sqrt{\tfrac{2j+1}{4\pi}} \textstyle\sum_p (-1)^p \frac{[(j+m)!(j-m)!]^{1/2} j!}{(j+m-p)!(j-p)!p!(j-m)!} \\ \times (q_0 + iq_3)^{j+m-p} (q_0 - iq_3)^{j-p} (q_2 + iq_1)^{p-m} (q_2 - iq_1)^p. \end{aligned} \tag{3.55}$$

Using the normalization of the spherical harmonics and Eq. (3.52) one arrives at the following expression for the p-coefficients

$$p_{mn}^j(t) = 4\pi \langle Y_m^j[\mathbf{q}(t)] Y_n^{*j}[\mathbf{q}(0)]\rangle. \tag{3.56}$$

The following relations for the p-coefficients hold:

$$p_{mn}^j(0) = \delta_{mn}^j, \tag{3.57}$$

$$p_{mn}^{*j}(t) = p_{-m-n}^j(t) = p_{nm}^j(-t). \tag{3.58}$$

The coefficients δ_{mn}^j are the components of the $(2j+1)\times(2j+1)$ unit matrix. The initial value of the the p-coefficients is an immediate consequence of definition

(3.46) and $p(\mathbf{\Omega}, 0|\mathbf{0}, 0) = \delta(\mathbf{\Omega})$. Eq. (3.58) follows from the symmetry of the Wigner functions and the symmetry of classical time correlation functions.

Since measurable quantities must be real it follows from (3.58) that only p-coefficients with $m = n = 0$ can be directly measured. $p^1_{00}(t)$ is measured by infrared spectroscopy (dipole-dipole correlation function) and $p^2_{00}(t)$ by relaxation NMR experiments. Here one measures in most cases the integral over $p^2_{00}(t)$.

3.9 Memory functions

Memory functions have been used for a long time in theoretical statistical physics to describe the time dependence of autocorrelation functions, Nevertheless, the use of memory functions in the context of Molecular Dynamics simulations has been hindered by the lack of a suitable numerical algorithm for their calculation. Such an algorithm has been published recently [33] and is now implemented in nMOLDYN. The key point is that a reliable estimates for memory functions can be obtained by assuming an Autoregressive (AR) model for the underlying stochastic process and not for the memory function itself (see Section 4.4).

Memory function equation of VACF In the context of VACF memory function one usually considers the normalized VACF which is defined as

$$\psi(t) := \frac{\langle v(t)v(0)\rangle}{\langle v^2(0)\rangle}. \tag{3.59}$$

Here $v(t)$ is the x-, y-, or z-component of the velocity of a 'tagged' atom. The memory function $\xi(t)$ of $\psi(t)$ is defined by the relation

$$\frac{d}{dt}\psi(t) = -\int_0^t d\tau\, \xi(t-\tau)\psi(\tau). \tag{3.60}$$

Eq. (3.60) is called the *memory function equation* (ME). For its numerical calculation, the dynamical variable under consideration , the velocity of a fluid particle, is considered as a discrete "signal", $v(n) = v(n\Delta t)$ which is modeled by an autoregressive stochastic process of order P. An overview of this algohritm is done in section 4.4 The reader is referred to Ref. [33] for more details.

Memory function equation of Coherent scattering Another memory function that can be calculated by nMOLDYN is the memory function related to the coherent intermediate scattering function. It is defined through the corresponding memory function equation

$$\partial_t \mathcal{F}_{coh}(\mathbf{q}, t) = -\int_0^t d\tau\, \xi(\mathbf{q}, t-\tau)\mathcal{F}_{coh}(\mathbf{q}, \tau). \tag{3.61}$$

The memory function $\xi(\mathbf{q}, t)$, which depends on time as well as on q, permits the analysis of memory effects on different length scales. From a numerical point

of view the calculation of the memory function equation relevant to the coherent intermediate scattering function is completely analogous to the case of the VACF memory function, the discrete time signal being here

$$\sum_{\alpha=1}^{N} b_{\alpha,coh} \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(t)]. \tag{3.62}$$

See section 4.4 for more details on the numerical algorithm.

Memory function equation of Incoherent scattering nMOLDYN allows one to calculate the memory function related to the incoherent intermediate scattering function as well. It is defined through the corresponding memory function equation

$$\partial_t \mathcal{F}_{inc}(\mathbf{q}, t) = -\int_0^t d\tau\, \xi(\mathbf{q}, t-\tau)\mathcal{F}_{inc}(\mathbf{q}, \tau). \tag{3.63}$$

The memory function $\xi(\mathbf{q}, \mathbf{t})$, which depends on q as well as on time, permits the analysis of memory effects on different length scales. As in the previuos cases, the numerical calculation of the memory function equation relevant to the incoherent intermediate scattering function is based on the Autoregressive model, the discrete time signal being here

$$\sum_{\alpha=1}^{N} b_{\alpha,inc} \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(t)]. \tag{3.64}$$

See section 4.4 for more details on the numerical algorithm.

Chapter 4

Algorithms

4.1 Computation of time correlation functions

In Section 3 it has been outlined that most of the quantities which can be extracted from MD simulations are time correlation functions. Correlation functions of discrete time series can be efficiently calculated by using the Fast Fourier Transform (FFT) [40]. The so-called Fast Correlation Algorithm (FCA) allows the number of multiplications (complexity) to be reduced from $\propto N_t^2$ to $\propto N_t \log_2(N_t)$. In *n*MOLDYN all time correlation functions are computed using the FCA method which will be outlined in the following. We will also briefly comment on spectral smoothing of Fourier transformed correlation functions.

We consider two time series

$$a(k \cdot \Delta t), \qquad b(k \cdot \Delta t), \qquad k = 0 \ldots N_t - 1, \tag{4.1}$$

of length $T = (N_t - 1) \cdot \Delta t$ which are to be correlated. In the following the shorthands $a(k)$ and $b(k)$ will be used. The discrete correlation function of $a(k)$ and $b(k)$ is defined as

$$c_{ab}(m) \doteq \begin{cases} \frac{1}{N_t - m} \sum_{k=0}^{N_t - m - 1} a^*(k) b(k + m), & m = 0 \ldots N_t - 1, \\ \\ \frac{1}{N_t - |m|} \sum_{k=|m|}^{N_t - 1} a^*(k) b(k - |m|), & m = -(N_t - 1) \ldots -1. \end{cases} \tag{4.2}$$

The prefactors in front of the sums ensure the proper normalization of the individual channels, $m = -(N_t - 1) \ldots N_t - 1$. The asterisk denotes a complex conjugate. According to (4.2), $c_{ab}(m)$ has $2N_t - 1$ data points and obeys the symmetry relation

$$c_{ab}(m) = c_{ba}^*(-m). \tag{4.3}$$

In case that $a(k)$ and $b(k)$ are identical, the corresponding correlation function $c_{aa}(m)$ is called an *auto*correlation function. We define now the extended,

periodic time series

$$A(k) = \begin{cases} a(k) & k = 0 \dots N_t - 1 \\ 0 & k = N_t \dots 2N_t - 1 \end{cases}, \tag{4.4}$$

$$B(k) = \begin{cases} b(k) & k = 0 \dots N_t - 1 \\ 0 & k = N_t \dots 2N_t - 1 \end{cases}, \tag{4.5}$$

which have the period $2N_t$,

$$A(k) = A(k+m\cdot 2N_t), \qquad B(k) = B(k+m\cdot 2N_t), \qquad m = 0, \pm 1, \pm 2, \dots . \tag{4.6}$$

The discrete, cyclic correlation of $A(k)$ and $B(k)$ is defined as

$$S_{AB}(m) = \sum_{k=0}^{2N_t-1} A^*(k) B(k+m). \tag{4.7}$$

It is easy to see that

$$c_{ab}(m) = \frac{1}{N_t - |m|} S_{AB}(m), \qquad -(N_t - 1) \le m \le N_t - 1. \tag{4.8}$$

Using the correlation theorem of discrete periodic functions [40], $S_{AB}(m)$ can be written as

$$S_{AB}(m) = \frac{1}{2N_t} \sum_{n=0}^{2N_t-1} \exp\left[2\pi i \left(\frac{mn}{2N_t}\right)\right] \tilde{A}^*\left(\frac{n}{2N_t}\right) \tilde{B}\left(\frac{n}{2N_t}\right) \tag{4.9}$$

where $\tilde{A}\left(\frac{n}{2N_t}\right)$ and $\tilde{B}\left(\frac{n}{2N_t}\right)$ are the discrete Fourier transforms of $A(k)$ and $B(k)$, respectively:

$$\tilde{A}\left(\frac{n}{2N_t}\right) = \sum_{k=0}^{2N_t-1} \exp\left[-2\pi i \left(\frac{nk}{2N_t}\right)\right] A(k), \tag{4.10}$$

$$\tilde{B}\left(\frac{n}{2N_t}\right) = \sum_{k=0}^{2N_t-1} \exp\left[-2\pi i \left(\frac{nk}{2N_t}\right)\right] B(k). \tag{4.11}$$

If the Fourier transforms of the signals $A(k)$ and $B(k)$ as well as the inverse transform in (4.9) are computed by FFT, $S_{AB}(m)$ can be computed by $\propto N_t \log_2(N_t)$ instead of $\propto N_t^2$ multiplications. It is sometimes said that the FFT method induces spurious correlations. We emphasize that this is only the case if the time series $a(k)$ and $b(k)$ are not properly extended, as indicated in Eqs. (4.4) and (4.5). The FFT method and the direct scheme (4.2) give, apart from round-off errors, *identical results*.

In many cases not only the computation of a correlation function is required, but also the computation of its Fourier spectrum. In principle one could use the product $\tilde{A}^*\left(\frac{n}{2N_t}\right) \tilde{B}\left(\frac{n}{2N_t}\right)$ which is already available as an intermediate step

in the computation of $S_{AB}(m)$ according to (4.9). This would, however, not be a good estimate for the spectrum of $c_{ab}(m)$ [41]. In nMOLDYN all spectra are smoothed by applying a window in the time domain [41]:

$$P_{ab}\left(\frac{n}{2N_t}\right) = \Delta t \cdot \sum_{m=-(N_t-1)}^{N_t-1} \exp\left[-2\pi i\left(\frac{nm}{2N_t}\right)\right] W(m)\,\frac{1}{N-|m|} S_{AB}(m). \tag{4.12}$$

The time step Δt in front of the sum yields the proper normalization of the spectrum. In nMOLDYN a Gaussian window [38] is used:

$$W(m) = \exp\left[-\frac{1}{2}\left(\alpha\frac{|m|}{N_t-1}\right)^2\right], \qquad m = -(N_t-1)\ldots N_t-1. \tag{4.13}$$

Its widths in the time and frequency domain are $\sigma_t = \alpha/T$ and $\sigma_\nu = 1/(2\pi\sigma_t)$, respectively. We recall that $T = (N_t-1)\cdot\Delta t$ is the length of the simulation. σ_ν corresponds to the width of the resolution function of the Fourier spectrum.

4.2 Mean-square displacements

The FCA-method described above may also be used to compute mean-square displacements [42]. In the discrete case the mean-square displacement of a particle is given by

$$\Delta^2(m) = \frac{1}{N_t-m}\sum_{k=0}^{N_t-m-1}[\mathbf{r}(k+m)-\mathbf{r}(k)]^2, \qquad m = 0\ldots N_t-1, \tag{4.14}$$

where $\mathbf{r}(k)$ is the particle trajectory. We define now the auxiliary function

$$S(m) \doteq \sum_{k=0}^{N_t-m-1}[\mathbf{r}(k+m)-\mathbf{r}(k)]^2, \qquad m = 0\ldots N_t-1, \tag{4.15}$$

which is split as follows:

$$S(m) = S_{AA+BB}(m) - 2S_{AB}(m), \tag{4.16}$$

$$S_{AA+BB}(m) = \sum_{k=0}^{N_t-m-1}[\mathbf{r}^2(k+m)+\mathbf{r}^2(k)], \tag{4.17}$$

$$S_{AB}(m) = \sum_{k=0}^{N_t-m-1}\mathbf{r}(k)\cdot\mathbf{r}(k+m). \tag{4.18}$$

The function $S_{AB}(m)$ can be computed using the FCA method described in Section 4.1. For $S_{AA+BB}(m)$ the following recursion relation holds:

$$S_{AA+BB}(m) = S_{AA+BB}(m-1) - \mathbf{r}^2(m-1) - \mathbf{r}^2(N_t-m), \tag{4.19}$$

$$S_{AA+BB}(0) = \sum_{k=0}^{N_t-1}\mathbf{r}^2(k). \tag{4.20}$$

This allows one to construct the following efficient scheme for the computation of mean-square displacements:

1. Compute $DSQ(k) = \mathbf{r}^2(k), \qquad k = 0 \ldots N_t - 1; \qquad DSQ(-1) = DSQ(N_t) = 0.$
2. Compute $SUMSQ = 2 \cdot \sum_{k=0}^{N_t-1} DSQ(k)$.
3. Compute $S_{AB}(m)$ using the FFT method.
4. Compute mean-square displacement $MSD(m)$ in the following loop:

$$\begin{aligned} SUMSQ &\leftarrow SUMSQ - DSQ(m-1) - DSQ(N_t - m) \\ MSD(m) &\leftarrow (SUMSQ - 2 \cdot S_{AB}(m)/(N_t - m) \end{aligned}$$

m running from 0 to $N_t - 1$

It should be noted that the efficiency of this algorithm is the same as for the FCA computation of time correlation functions since the number of operations in step (1), (2), and (4) grows linearly with N_t.

4.3 Rigid-body fits

We discuss now briefly how rigid body motions, i.e. global translations and rotations of molecules or subunits of complex molecules, can be extracted from a molecular dynamics trajectory. A more detailed presentation is given in [43]. We define an optimal rigid-body trajectory in the following way: For each time frame of the trajectory the atomic positions of a rigid reference structure, defined by the three cartesian components of its centroid (e.g. the center of mass) and three angles, are as close as possible to the atomic positions of the corresonding structure in the MD configuration. Here 'as close as possible' means as close as possible in a least-squares sense.

Optimal superposition. We consider a given time frame in which the atomic positions of a (sub)molecule are given by $\mathbf{x}_\alpha$, $\alpha = 1 \ldots N$. The corresponding positions in the reference structure are denoted as $\mathbf{x}_\alpha^{(0)}$, $\alpha = 1 \ldots N$. For both the given structure and the reference structure we introduce the yet undetermined centroids $\mathbf{X}$ and $\mathbf{X}^{(0)}$, respectively, and define the deviation

$$\mathbf{\Delta}_\alpha \doteq \mathbf{D}(\mathbf{q}) \left[\mathbf{x}_\alpha^{(0)} - \mathbf{X}^{(0)} \right] - \left[\mathbf{x}_\alpha - \mathbf{X} \right]. \tag{4.21}$$

Here $\mathbf{D}(\mathbf{q})$ is a rotation matrix which depends on also yet undetermined angular coordinates which we chose to be *quaternion parameters*, abbreviated as vector $\mathbf{q} = (q_0, q_1, q_2, q_3)$. The quaternion parameters fulfill the normalization condition $\mathbf{q} \cdot \mathbf{q} = 1$ [26]. The target function to be minimized is now defined as

$$m(\mathbf{q}; \mathbf{X}, \mathbf{X}^{(0)}) = \sum_\alpha w_\alpha |\mathbf{\Delta}|_\alpha^2. \tag{4.22}$$

The w_α are normalized positive weights, $\sum_\alpha w_\alpha = 1$. The minimization with respect to the centroids is decoupled from the minimization with respect to the quaternion parameters and yields

$$\mathbf{X} = \sum_\alpha w_\alpha \, \mathbf{x}_\alpha, \tag{4.23}$$

$$\mathbf{X}^{(0)} = \sum_\alpha w_\alpha \, \mathbf{x}_\alpha^{(0)}. \tag{4.24}$$

We are now left with a minimization problem for the rotational part which can be written as

$$m(\mathbf{q}) = \sum_\alpha w_\alpha \left[\mathbf{D}(\mathbf{q}) \mathbf{r}_\alpha^{(0)} - \mathbf{r}_\alpha \right]^2 \stackrel{!}{=} Min. \tag{4.25}$$

The relative position vectors

$$\mathbf{r}_\alpha = \mathbf{x}_\alpha - \mathbf{X}, \tag{4.26}$$

$$\mathbf{r}_\alpha^{(0)} = \mathbf{x}_\alpha^{(0)} - \mathbf{X}^{(0)}, \tag{4.27}$$

are fixed and the rotation matrix reads [26]

$$\mathbf{D}(\mathbf{q}) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(-q_0 q_3 + q_1 q_2) & 2(q_0 q_2 + q_1 q_3) \\ 2(q_0 q_3 + q_1 q_2) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(-q_0 q_1 + q_2 q_3) \\ 2(-q_0 q_2 + q_1 q_3) & 2(q_0 q_1 + q_2 q_3) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{pmatrix}. \tag{4.28}$$

Quaternions and rotations. The rotational minimization problem can be elegantly solved by using quaternion algebra. Quaternions are so-called hypercomplex numbers, having a real unit, **1**, and three imaginary units, **I**, **J**, and **K**. Since **IJ** = **K** (cyclic), quaternion multiplication is not commutative. A possible matrix representation of an arbitrary quaternion,

$$\mathbf{A} = a_0 \cdot \mathbf{1} + a_1 \cdot \mathbf{I} + a_2 \cdot \mathbf{J} + a_3 \cdot \mathbf{K}, \tag{4.29}$$

reads

$$\mathbf{A} = \begin{pmatrix} a_0 & -a_1 & -a_2 & -a_3 \\ a_1 & a_0 & -a_3 & a_2 \\ a_2 & a_3 & a_0 & -a_1 \\ a_3 & -a_2 & a_1 & a_0 \end{pmatrix}. \tag{4.30}$$

The components a_ν are real numbers. Similarly as normal complex numbers allow one to represent rotations in a plane, quaternions allow one to represent rotations in space. Consider the quaternion representation of a vector **r**, which is given by

$$\mathbf{R} = x \cdot \mathbf{I} + y \cdot \mathbf{J} + z \cdot \mathbf{K}, \tag{4.31}$$

and perform the operation

$$\mathbf{R}' = \mathbf{Q}\mathbf{R}\mathbf{Q}^T, \tag{4.32}$$

where $\mathbf{Q}$ is a normalized quaternion,

$$\|\mathbf{Q}\|^2 \doteq q_0^2 + q_1^2 + q_2^2 + q_3^2 = \frac{1}{4} tr\{\mathbf{Q}^T\mathbf{Q}\} = 1. \tag{4.33}$$

The symbol tr stands for 'trace'. We note that a normalized quaternion is represented by an *orthogonal* 4×4 matrix. $\mathbf{R}'$ may then be written as

$$\mathbf{R}' = x' \cdot \mathbf{I} + y' \cdot \mathbf{J} + z' \cdot \mathbf{K}, \tag{4.34}$$

where the components x', y', z', abbreviated as $\mathbf{r}'$, are given by

$$\mathbf{r}' = \mathbf{D}(\mathbf{q})\mathbf{r}. \tag{4.35}$$

The matrix $\mathbf{D}(\mathbf{q})$ is the rotation matrix defined in (4.28).

Solution of the minimization problem. In quaternion algebra, the rotational minimization problem may now be phrased as follows:

$$m(\mathbf{q}) = \sum_\alpha w_\alpha \|\mathbf{Q}\mathbf{R}_\alpha^{(0)}\mathbf{Q}^T - \mathbf{R}_\alpha\|^2 \stackrel{!}{=} Min. \tag{4.36}$$

Since the matrix $\mathbf{Q}$ representing a normalized quaternion is orthogonal this may also be written as

$$m(\mathbf{q}) = \sum_\alpha w_\alpha \|\mathbf{Q}\mathbf{R}_\alpha^{(0)} - \mathbf{R}_\alpha\mathbf{Q}\|^2. \stackrel{!}{=} Min. \tag{4.37}$$

This follows from the simple fact that $\|\mathbf{A}\| = \|\mathbf{A}\mathbf{Q}\|$, if $\mathbf{Q}$ is normalized. Eq. (4.37) shows that the target function to be minimized can be written as a simple quadratic form in the quaternion parameters [43],

$$m(\mathbf{q}) = \mathbf{q} \cdot \mathbf{M}\mathbf{q}, \tag{4.38}$$

$$\mathbf{M} = \sum_\alpha w_\alpha \mathbf{M}_\alpha. \tag{4.39}$$

The matrices $\mathbf{M}_\alpha$ are positive semi-definite matrices depending on the positions $\mathbf{r}_\alpha$ and $\mathbf{r}_\alpha^{(0)}$:

$$\left.\begin{aligned}
M_{\alpha,11} &= x_\alpha^2 + y_\alpha^2 + z_\alpha^2 + x_{0\alpha}^2 + y_{0\alpha}^2 + z_{0\alpha}^2 - 2x_\alpha x_{0\alpha} - 2y_\alpha y_{0\alpha} - 2z_\alpha z_{0\alpha} \\
M_{\alpha,12} &= 2(y_\alpha z_{0\alpha} - z_\alpha y_{0\alpha}) \\
M_{\alpha,13} &= 2(-x_\alpha z_{0\alpha} + z_\alpha x_{0\alpha}) \\
M_{\alpha,14} &= 2(x_\alpha y_{0\alpha} - y_\alpha x_{0\alpha}) \\
M_{\alpha,22} &= x_\alpha^2 + y_\alpha^2 + z_\alpha^2 + x_{0\alpha}^2 + y_{0\alpha}^2 + z_{0\alpha}^2 - 2x_\alpha x_{0\alpha} + 2y_\alpha y_{0\alpha} + 2z_\alpha z_{0\alpha} \\
M_{\alpha,23} &= -2(x_\alpha y_{0\alpha} + y_\alpha x_{0\alpha}) \\
M_{\alpha,24} &= -2(x_\alpha z_{0\alpha} + z_\alpha x_{0\alpha}) \\
M_{\alpha,33} &= x_\alpha^2 + y_\alpha^2 + z_\alpha^2 + x_{0\alpha}^2 + y_{0\alpha}^2 + z_{0\alpha}^2 + 2x_\alpha x_{0\alpha} - 2y_\alpha y_{0\alpha} + 2z_\alpha z_{0\alpha} \\
M_{\alpha,44} &= -2(y_\alpha z_{0\alpha} + z_\alpha y_{0\alpha}) \\
M_{\alpha,44} &= x_\alpha^2 + y_\alpha^2 + z_\alpha^2 + x_{0\alpha}^2 + y_{0\alpha}^2 + z_{0\alpha}^2 + 2x_\alpha x_{0\alpha} + 2y_\alpha y_{0\alpha} - 2z_\alpha z_{0\alpha}
\end{aligned}\right\} \tag{4.40}$$

The rotational fit is now reduced to the problem of finding the minimum of a quadratic form with the constraint that the quaternion to be determined must be normalized. Using the method of Lagrange multipliers to account for the normalization constraint we have

$$m'(\mathbf{q},\lambda) = \mathbf{q}\cdot\mathbf{M}\mathbf{q} - \lambda(\mathbf{q}\cdot\mathbf{q}-1) \stackrel{!}{=} Min. \tag{4.41}$$

This leads immediately to the eigenvalue problem

$$\begin{aligned} \mathbf{M}\mathbf{q} &= \lambda\mathbf{q}, & (4.42)\\ \mathbf{q}\cdot\mathbf{q} &= 1. & (4.43) \end{aligned}$$

Now any normalized eigenvector $\mathbf{q}$ fulfills the relation $\lambda = \mathbf{q}\cdot\mathbf{M}\mathbf{q} \equiv m(\mathbf{q})$. Therefore the eigenvector belonging to the smallest eigenvalue, λ_{min}, is the desired solution. At the same time λ_{min} gives the average error per atom.

4.4 Autoregressive (AR) process

To compute the memory function $\xi(t)$ from a discrete "signal" $x(n) \equiv x(n\Delta t)$ the latter is modelled by an *autoregressive stochastic process* of order P [44, 45],

$$x(t) = \sum_{n=1}^{P} a_n^{(P)} x(t-n\Delta t) + \epsilon_P(t). \tag{4.44}$$

Here $\epsilon_P(t)$ is *white noise* with zero mean and amplitude σ_P. The coeffients $\{a_n^{(P)}\}$ are fitted to the discrete "signal" using Burg's algorithm [46, 47], and σ_P is given by

$$\sigma_P^2 = r(0) - \sum_{n=1}^{P} a_n^{(P)}(n\Delta t), \tag{4.45}$$

where $r(t)$ is the autocorrelation function of $x(t)$

$$r(t) := \langle x(t)x(0)\rangle. \tag{4.46}$$

In all following calculations nMOLDYN works with a set of coefficients $\{a_n\}$ which has been averaged over all selected atoms and the three Cartesian coordinates.

VACF within the AR model The autocorrelation function $r(t)$ introduced in the previous Section is here the *normalized* velocity autocorrelation function (VACF)

$$\psi(t) := \frac{\langle v(t)v(0)\rangle}{\langle v^2(0)\rangle} \tag{4.47}$$

hence $r(0) = \psi(0) = 1$. Within the AR-model the z-transform of the VACF has the form

$$\Psi^{(AR)}(z) = \frac{1}{a_P^{(P)}} \frac{-z^P\sigma_P^2}{\prod_{k=1}^{P}(z-z_k)\prod_{l=1}^{P}(z-z_l^{-1})}. \tag{4.48}$$

Here the $\{z_k\}$ are the zeros of

$$p(z) = z^P - \sum_{k=1}^{P} a_k^{(P)} z^{P-k}. \tag{4.49}$$

We recall that the z-transform of an arbitrary discrete function $f(n)$ is given by $F(z) = \sum_{n=-\infty}^{+\infty} f(n) z^{-n}$, and the inverse transform by $f(n) = \frac{1}{2\pi i} \oint_C dz\, z^{n-1} F(z)$. Applying the inverse z-transform to (4.48) yields

$$\psi^{(AR)}(n) = \sum_{j=1}^{P} \beta_j z_j^{|n|}, \tag{4.50}$$

where the coefficients β_j are given by

$$\beta_j = \frac{1}{a_P} \frac{-z_j^{P-1} \sigma_P^2}{\prod_{k=1, k\neq j}^{P} (z_j - z_k) \prod_{l=1}^{P} (z_j - z_l^{-1})}. \tag{4.51}$$

Note that $\psi^{(AR)}(n)$ has a multiexponential form, and that the stability criterion

$$|z_j| < 1, \quad j = 1, \ldots, P, \tag{4.52}$$

must be fulfilled. This is guaranteed by the Burg-algorithm [46, 47].

Density of states within the AR model Evaluating $\Psi^{(AR)}(z)$ as given by (4.48) at $z = \exp(i\omega\Delta t)$ yields the density of states within the AR model:

$$G^{(AR)}(\omega) = \frac{\Delta t}{2} \frac{k_B T}{M} \Psi^{(AR)} \left(\exp[i\omega\Delta t]\right). \tag{4.53}$$

Here M is the mass of the tagged atom, k_B is the Boltzmann constant, and T the temperature. Note that the VACF and the density of states within the AR model are entirely determined by the coefficients $a_n^{(P)}$.

Discrete memory function of the VACF within the AR model Starting from the memory function equation of the VACF (Eq. 3.60), the first step towards a numerical computation of the memory function consists in discretizing Eq. 3.60

$$\frac{\psi(n+1) - \psi(n)}{\Delta t} = -\sum_{k=0}^{n-1} \Delta t\, \xi(n-k)\psi(k), \tag{4.54}$$

Eq. (4.54) is now subjected to a *one-sided* z-transform. Using that

$$Z_> \{f(n+1) - f(n)\} = z F_>(z) - z f(0) \tag{4.55}$$

for any discrete function $f(n)$ whose one-sided z-transform exists, one obtains from (4.54)

$$\Xi_>(z) = \frac{1}{\Delta t^2} \left(\frac{z}{\Psi_>(z)} + 1 - z \right), \tag{4.56}$$

using that $\psi(0) = 1$. The one-sided z-transform of an arbitrary discrete function $f(n)$ is defined as $F_>(z) = \sum_{n=0}^{\infty} f(n) z^{-n}$. Here it has been used that the one-sided z-transform of the discrete convolution integral is just the product $\Xi_>(z)\Psi_>(z)$. Inserting the definition of the one-sided z-transform for $\Xi_>(z)$ and $\Psi_>(z)$, this equation can be rewritten as

$$\sum_{j=0}^{\infty} \xi(j)\, z^{-j} = \frac{1}{\Delta t^2} \frac{\sum_{j=0}^{\infty} \left[\psi(j) - \psi(j+1)\right] z^{-j}}{\sum_{j=0}^{\infty} \psi(j)\, z^{-j}}. \tag{4.57}$$

Note that the term proportional to z cancels out. The time dependent memory function is, in principle, obtained by comparing the coefficients of the series on the *lhs* and the *rhs* of Eq. (4.57). To construct a numerical method, we replace the series by polynomials of order N, where $t = N\Delta t$ defines the time window for the memory function to be computed. After this first step a polynomial division is performed on the *rhs* of Eq. (4.57), and after a subsequent multiplication of both sides with z^{-N} one obtains the time dependent memory function, $\xi(j)$, by comparison of coefficients,

$$\begin{aligned} \frac{z^{-N}}{\Delta t^2} \frac{\sum_{j=0}^{N} \left[\psi(j) - \psi(j+1)\right] z^{N-j}}{\sum_{j=0}^{N} \psi(j)\, z^{-j}} &= z^{-N} \left(\sum_{j=0}^{N} c_j\, z^{N-j} + R \right) \\ &= \sum_{j=0}^{N} \xi(j)\, z^{-j}. \end{aligned} \tag{4.58}$$

Within nMoldyn $\psi(n)$ is replaced by the autocorrelation function calculated in the framework of the autoregressive model, $\psi^{(AR)}(n)$, as in Eqs. 4.50 and 4.51. The coefficients c_j are obtained by polynomial division and R is a rest which does not contain information on the memory function within the time interval $t \in [0, N\Delta t]$. The discrete memory function is therefore given by $\xi(j) = c_{N-j}$.

A remark concerning the discretization scheme (4.54) is in place here. The discrete convolution sum is effectively a first order approximation of the convolution integral. More sophisticated approximations could be used, but they would lead to less convenient expressions upon z-transformation. Correspondingly, we have chosen a first order approximation for the differentiation on the left-hand side of (3.60). In this way the first order (integro-) differential equation (3.60) is transformed into the first order difference equation (4.54).

However, this simple discretization scheme together with the use of the one-sided z-transform leads to a significant error in $\xi(0)$. It is clear from Eq. (3.60) that due to the symmetry of the autocorrelation function ($\psi(t) = \psi(-t)$), the derivative $d\psi/dt$ should vanish at $t = 0$. However, in the discretized version it is approximated by a forward difference that is always negative. A higher-order calculation shows that the estimate for $\xi(0)$ that results from the procedure described above should be doubled.

MSD within the AR model The relation between the VACF and the MSD reads

$$MSD(t) = \langle [x(t) - x(0)]^2 \rangle = 2 \int_0^t d\tau \, (t - \tau) C_{vv}(t) \tag{4.59}$$

By discretizing the above equation one obtains

$$MSD(n) = 2 \sum_{k=0}^{n} \Delta t^2 (n - k) C_{vv}(k). \tag{4.60}$$

Using

$$f_1(n) = \Theta(n) \cdot n f_2(n) = \Theta(n) \cdot C_{vv}(n) \tag{4.61}$$

into Eq. 4.60, gives

$$MSD(n) = 2 \sum_{k=-\infty}^{+\infty} \Delta t^2 f_1(n - k) f_2(k) \tag{4.62}$$

Making use of the one-side z-transform (equivalent to the Laplace transform for discrete functions), we obtain

$$MSD^{>}(z) = 2 F_1^{>} F_2^{>} \Delta t^2, \tag{4.63}$$

where

$$F_1^{>} = \frac{z}{(z-1)^2}. \tag{4.64}$$

Introducing Eq. 4.64 into Eq. 4.63 yields

$$MSD^{>}(z) = 2 \frac{z \Delta t^2}{(z-1)^2} C_{vv}^{>}(z) \tag{4.65}$$

and its inverse z-transform reads

$$MSD(n) = 2\Delta t^2 \cdot \frac{1}{2\pi i} \oint dz z^{n-1} \cdot \frac{z}{(z-1)^2} \cdot C_{vv}^{>}(z). \tag{4.66}$$

Using the expression of the non-normalized $C_{vv}^{\rangle}(z)$ obtained in the framework of the AR model

$$C_{vv}^{>}(z) = \sum_{j=1}^{P} \beta_j \frac{z}{z - z_j} \cdot \langle v^2 \rangle \tag{4.67}$$

one finds the expression of MSD within the same AR model

$$MSD_{AR}(n) = 2\Delta t^2 \langle v^2 \rangle \sum_{j=1}^{P} \beta_j \frac{1}{2\pi i} \oint dz \frac{z^{n+1}}{(z-1)^2} \cdot \frac{1}{z - z_j} \tag{4.68}$$

$$= 2\Delta t^2 \langle v^2 \rangle \sum_{j=1}^{P} \beta_j \{ \frac{n}{1 - z_j} - \frac{z_j}{(1 - z_j)^2} + \frac{z_j^{n+1}}{(1 - z_j)^2} \}. \tag{4.69}$$

In the limit of $n \to +\infty$, the above expression reads

$$MSD_{AR}(n) \overset{n\to+\infty}{\simeq} 2Dn\Delta t \; ; \; D = \Delta t \langle v^2 \rangle \sum_{j=1}^{P} \frac{\beta_j}{1 - z_j} = \frac{\langle v^2 \rangle}{\gamma} \tag{4.70}$$

, which allows one to compute the MSD within the AR model from the poles and the β_j coefficients of the non-normalized VACF.

Friction coefficient within the AR model The friction coefficient is defined as the integral over the memory function. In the discrete case we write

$$\xi_0 := \sum_{n=0}^{\infty} \Delta t \, \xi(n) = \Delta t \, \Xi_>(1). \tag{4.71}$$

As shown in [33], the AR model allows us to express $\Psi_>(z)$ as

$$\Psi_>^{(AR)}(z) = \sum_{n=0}^{\infty} \psi^{(AR)}(n) z^{-n} = \sum_{j=1}^{P} \beta_j \frac{z_j}{z - z_j}, \tag{4.72}$$

where the coefficients β_j are given by eq. (4.51), and the roots z_j must fulfill the stability criterion (4.52). Inserting (4.72) into (4.56) yields $\Xi_>^{(AR)}(z)$, the z-transform of the discrete memory function within the AR model,

$$\Xi_>^{(AR)}(z) = \frac{1}{\Delta t^2} \left(\frac{z}{\Psi_>^{(AR)}(z)} + 1 - z \right). \tag{4.73}$$

Using (4.73) we obtain thus within the AR model

$$\xi_0^{(AR)} = \frac{1}{\Delta t} \frac{1}{\sum_{j=1}^{P} \beta_j \frac{1}{1 - z_j}}. \tag{4.74}$$

This shows that ξ_0 can be obtained from the zeros z_j of the characteristic polynomial $p(z)$, defined in (4.49).

F_{inc} and F_{coh} within the AR model The computation of F_{inc} and F_{coh} within the AR model is carried out using same algorithm emploied for the calculation of the VACF within the AR model Here for each q values, the discrete time signal is

$$\sum_{\alpha=1}^{N} b_{\alpha,inc} \exp[-i\mathbf{q} \cdot \mathbf{R}_\alpha(t)]. \tag{4.75}$$

For each q-vector one obtains a set of P coefficients β_j.

Discrete memory function of the F_{inc} and F_{coh} within the AR model For the computation of the discrete memory functions of F_{inc} and F_{coh}, nMoldyn uses the same algorithm emploied for the computation of the discrete memory function of the VACF Since the memory function $\xi(\mathbf{q}, \mathbf{t})$ depends on q as well as on time, for each q-vector one obtains a set of P complex coefficients c_j.

Bibliography

[1] S.W. Lovesey, 'Theory of Neutron Scattering from Condensed Matter', Vol. 1, Clarendon Press, Oxford, 1984.

[2] M. Bee, 'Quasielastic Neutron Scattering: Principles and Applications in Solid State Chemistry, Biology, and Materials Science', Adam Hilger, Bristol, 1988.

[3] G.R. Kneller, *Mol. Phys.* in press.

[4] G.R. Kneller and A. Geiger, *Molecular Physics* **68**(2), 487-498 (1989).

[5] G.R. Kneller and A. Geiger, *Molecular Physics* **70**(3), 465-483 (1990).

[6] H.J. Boehm and R. Ahlrichs, *Mol. Phys.* **54**(6), 1261-1274 (1985).

[7] J. Anderson, J.J. Ullo, S. Yip, *J. Chem. Phys.* **86**(7), 1987.

[8] G.R. Kneller, J.C. Smith, S. Cusack and W. Doster, *J. Chem. Phys.*, **97**(12), 8864 (1992).

[9] A.J. Dianoux, G.R. Kneller, J.L. Sauvageol, and J.C. Smith, *J. Chem. Phys.* **99**(7), 5586 (1993).

[10] G.van Rossum, et al. The Python web site, http://www.python.org/.

[11] D.Ascher, P.Dubois, K.Hinsen, J.Huguin, T.Oliphant, Numerical Python. Technical report UCRL-MA-128569, Lawrence Livermoore National Library, http://numpy.sourceforge.net

[12] The FFTW library. Technical Report. http://fftw.org.net

[13] http://my.unidata.ucar.edu/content/software/netcdf/software.html

[14] http://www.unidata.ucar.edu/software/netcdf/guide_12.html

[15] M. Abramowitz, I.A. Stegun, 'Handbook of Mathematical Functions', Dover, New York, 1972 p.914.

[16] http://sourcesup.cru.fr/projects/mmtk/

[17] R. Rew, G. Davis, S. Emmerson, H. Davies Netcd-fUser'sGuide for c, an interface for Data Access, version 3. http://www.unidata.ucar.edu/packages/netcdf/guidec, (1997).

[18] K.Hinsen, ScientificPython. http://dirac.cnrs-orleans.fr

[19] K.Hinsen *J.Comp.Chem* 2000, **21**, 79.

[20] OpenSource web site. http://www.opensource.org, (1997).

[21] L. van Hove, *Phys. Rev.* **95**(1), 249 (1954).

[22] P. Schofield, *Phys. Rev. Letters* **4**(5), 239 (1960).

[23] A. Rahman, K.S. Singwi, and A. Sjölander, *Phys. Rev.* **126**, 986 (1962).

[24] J.P. Boon and S. Yip, 'Molecular Hydrodynamics', McGraw-Hill, New York, 1980.

[25] J.-P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen, *J. Comput. Phys.* **23**, 327 (1977).

[26] S.L. Altmann, 'Rotations, Quaternions, and Double Groups', Clarendon Press, Oxford, 1986.

[27] G.R. Kneller and A. Geiger, *Molecular Simulation* **3**(5+6), 283-300 (1989).

[28] R.M. Lynden-Bell and A.J. Stone, *Molecular Simulation* **3**, 271-81 (1989)

[29] M.P. Allen and D.J. Tildesley, 'Computer Simulation of Liquids', Oxford University Press, Oxford, 1987

[30] A.R. Edmonds, 'Angular momentum in Quantum Mechanics', Princeton University Press, Princeton, New Jersey, 1957.

[31] D.M. Brink and G.R. Satchler, 'Angular Momentum', Clarendon Press, Oxford, 1968.

[32] M.E. Rose, 'Elementary Theory of Angular Momentum', John Wiley, New York, 1957.

[33] G.R.Kneller, K. Hinsen *J. Chem. Phys.* 2001, **115**, 11097

[34] http://www.cecill.info/

[35] http://www.python.org/download/

[36] http://www.unidata.ucar.edu/software/netcdf/

[37] http://sourcesup.cru.fr/projects/scientific-py/

[38] F.J. Harris, *Proc. IEEE* **66**(1), 51 (1978).

[39] T.Rog, K.Murzin, K.Hinsen, G.R.Kneller, J.Comp.Chem 2002, 24, 657

[40] E.O. Brigham, 'The Fast Fourier Transfrom', Prentice Hall, Englewood Cliffs (NJ) USA, 1974.

[41] A. Papoulis, 'Signal Analysis', McGraw-Hill, Singapore, 1984.

[42] G.R. Kneller, Technical Report Jül 2215, Forschungszentrum Jülich (ISSN 0366-0885), ZB Forschungszentrum Jülich, D-52425 Jülich, Germany.

[43] G.R. Kneller, *Molecular Simulation* **7**, 113 (1991).

[44] A. Papoulis, 'Probability, random variables, and Stochastic Processe', McGraw-Hill, New York, 1991;3rd ed.

[45] Makhoul, *J. Proc. IEEE* 1975, **63,** 561.

[46] J.Burg, Maximum Entropy Spectral Analysis. PhD thesis, Stanford University, Stanford, CA, 1975

[47] Makhoul, *J. IEEE Transactions on Acoustics, Speech, and Signal Processing.* **ASSP-25**, 1977,423.